A reliable Turing machine

Ilir Çapuni Peter Gács

June 5, 2015

Abstract

The title says it.

Contents

1 Introduction

1.1 To be written

1.2 Turing machines

Our contribution uses one of the standard definitions of a Turing machine.

A Turing machine M is defined by a tuple

$$(\Gamma, \Sigma, \tau, q_{\text{start}}, F).$$

Here, Γ is a finite set of *states*, Σ is a finite alphabet, and

$$\tau\colon \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{-1, 1\}$$

is the transition function. The tape alphabet Σ contains at least the distinguished symbols $\llcorner\!\lrcorner, 0, 1$ where $\llcorner\!\lrcorner$ is called the *blank symbol*. The state q_{start} is called the *starting state*, and there is a set F of *final states*.

The tape is blank at all but finitely many positions.

A *configuration* is a tuple

$$(q, A, h),$$

where $q \in \Gamma$ is the *current state*, $h \in \mathbb{Z}$ is the current *head position*, or *observed cell*, and $A \in \Sigma^{\mathbb{Z}}$ is the *tape content*: at position p, the tape contains the symbol $A(p)$. If $C = (q, A, h)$ is a configuration then we will write

$$C.\text{state} = q, \quad C.\text{pos} = h, \quad C.\text{tape} = A.$$

Here, A is also called the *tape configuration*. The cell at position h is the *current cell*. Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary.

The transition function τ tells us how to compute the next configuration from the present one. When the machine is in a state q, at tape position h, and observes tape cell with content a, then denoting

$$(a', q', j) = \tau(a, q),$$

it will change the state to q', change the tape content at position h to a', and move to tape position to $h + j$. For $q \in F$ we have $a' = a$, $q' \in F$.

Definition 1.1 (Fault) We say that a *fault* occurs at time t if the output (a', q', j) of the transition function at this time is replaced with some other value (which is then used to compute the next configuration). ⌟

1.3 Codes, and the result

For fault-tolerant computation, some redundant coding of the information is needed.

Definition 1.2 (Codes) Let Σ_1, Σ_2 be two finite alphabets. A *block code* is given by a positive integer Q – called the *block size* – and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. It is extended to strings by encoding each letter individually: $\psi_*(x_1, \ldots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n)$. ⌟

For ease of spelling out a result, let us consider only computations whose outcome is a single symbol, at tape position 0.

Theorem 1 *There is a Turing machine M_1 with a function $a \mapsto a.output$ defined on its alphabet, such that for any Turing machine G with alphabet Σ and state set Γ there are $0 \leq \varepsilon < 1$ and $\alpha_1, \alpha_2 > 0$ with the following property.*

For each input length $n = |x|$ a block code (φ_*, φ^*) of block size $Q = O((\log n)^{\alpha_1})$ can be constructed such that the following holds.

Let M_1 start its work from its initial state, and the initial tape configuration $\xi = (q_{\text{start}}, \varphi_*(x), 0)$. Assume further that during its operation, faults occur independently at random with probabilities $\leq \varepsilon$.

Suppose that on input x machine G reaches a final state at time t and writes value y at position 0 of the tape. Then denoting by $\eta(u)$ the configuration machine M_1 at time u, at any time t' after

$$t \cdot (\log t)^{\alpha_2},$$

we have $\eta(t').\text{tape}(0).output = y$ with probability at least $1 - O(\varepsilon)$.

We emphasize that the actual code φ of the construction will depend on n only in a simple way: it will be the "concatenation" of one and the same fixed-size code with itself, $O(\log \log n)$ times.

2 Overview of the construction

A Turing machine that simulates "reliably" any other Turing machine even when it is subjected to isolated bursts of faults of constant size, is given in [1]. By *reliably* we mean that the simulated computation can be decoded from the history of the simulating machine despite occasional damages.

2.1 Isolated bursts of faults

Let us give a brief overview of a machine M_1 that can withstand isolated bursts of faults, as most of its construction will be reused in the probabilistic setting.

Let us break up the task of error correction into several problems to be solved. The solution of one problem gives rise to another problem one, but the process converges.

Redundant information The tape information of the simulated Turing machine will be stored in a redundant form, more precisely in the form of a block code.

Redundant processing The block code will be decoded, the retrieved information will be processed, and the result recorded. To carry out all this in a way that limits the propagation of faults, the tape will be split into tracks that can be handled separately, and the major processing steps will be carried out three times within one work period.

Local repair All the above process must be able to recover from a local burst of faults. For this, it is organized into a rigid, locally checkable structure with the help of local addresses, and some other tools like sweeps and short switchbacks (zigzags). The major tool of local correction, the local healing procedure, turns out to be the most complex part of the construction.

Disturbed local repair A careful organization of the healing procedure makes sure that even if a new burst interrupts it (or jumps into its middle), soon one or two new invocations of it will finish the job (whenever needed).

Here is some more detail. Each tape cell of the simulated machine M_2 will be represented by a block of size Q of the simulating machine M_1 called a *colony*. Each step of M_2 will be simulated by a computation of M_1 called a *work period*. During this time, the head of M_1 makes a number of sweeps over the current colony, decodes the represented cell symbol and state, computes the new state, and transfers the necessary information to the neighbor colony that corresponds to the new position of the head of M_2.

In order to protect information from the propagation of errors, the tape of M_1 is subdivided into *tracks*: each track corresponds to a *field* of a cell symbol of M_1 viewed as a data record. Each stage of computation will be repeated three times. The results will be stored in separate tracks, and a final cell-by-cell majority vote will recover the result of the work period from them.

All this organization is controlled by a few key fields, for example a field called *cAddr* showing the position of each cell in the colony, and a field *cSw* showing the last sweep of the computation (along with its direction) that has been performed already. The technically most challenging part of the construction is the protection of this control information from bursts.

For example, a burst can reverse the head in the middle of a sweep. Our goal is that such structural disruptions be discovered locally, so we cannot allow the head to go far from the place where it was turned back. Therefore the head's movement will not be straight even during a single sweep: it will make frequent zigzags. This will trigger the healing procedure if for example a turn-back is detected.

It is a significant challenge that the healing procedure itself can be interrupted (or started) by a burst.

2.2 Hierarchical construction

In order to build a machine that can resist faults occurring independently of each other with some small probability, we take the approach suggested in [5], and implemented in [3] and [4] for the case of one-dimensional cellular automata, with some ideas from the tiling application of [2]. We will build a *hierarchy of simulations*: machine M_1 simulates machine M_2 which simulates machine M_3, and so on. For simplicity we assume all these machines have the same program, and all simulations have the same block size.

One cell of machine M_3 is simulated by one colony of machine M_2. Correspondingly, one cell of M_2 is simulated by one colony of machine M_1. So one cell of M_3 is simulated by Q^2 cells of M_1. Further, one step of machine M_3 is simulated by one work period of M_2 of, say, $O(Q^2)$ steps. One step of M_2 is simulated by one work period of M_1, so one step of M_3 is simulated by $O(Q^4)$ steps of M_1.

Per construction, machine M_1 can withstand bursts of faults whose size is $\leq \beta$ for some constant parameter β, that are separated by some $O(Q^2)$ fault-free steps. Machines M_2, M_3, ... have the same program, so it would be natural to expect that machine M_1 can withstand also some *additional*, larger bursts of size $\leq \beta Q$ if those are separated by at least $O(Q^4)$ steps.

But a new obstacle arises. On the first level, damage caused by a big burst spans several colonies. The repair mechanism of machine M_1 outlined in Section 2.1 above is too local to recover from such extensive damage. This cannot be allowed, since then the whole hierarchy would stop working. So we add a new mechanism to M_1 that, more modestly, will just try to restore a large enough portion of the tape, so it can go on with the simulation of M_2, even if all original information was lost. For this, M_1 may need to rewrite an area as large as a few colonies.

This will enable the low-level healing procedure of machine M_2 to restore eventually a higher-level order.

All machines above M_1 in the hierarchy are "virtual": the only hardware in the construction is machine M_1. Moreover, they will not be ordinary Turing machines, but *generalized* ones, with some new features that are not needed on the lowest level but seem necessary in a simulated Turing machine: for example they allow a positive distance between neighboring tape cells.

A tricky issue is "forced self-simulation": while we are constructing machine M_1 we want to give it the feature that it will simulate a machine M_2 that works just like M_1. The "forced" feature means that this simulation should work without any written program (that could be corrupted).

This will be achieved by a construction similar to the proof of the Kleene's fixed-point theorem (also called recursion theorem). We first fix a (simple) programming language to express the transition function of a Turing machine. We write an interpreter for it in this same language (just as compilers for the C language are sometimes written in C). The program of the transition function of M_2 (essentially the same as that of M_1) in this language, is a string that will be "hard-wired" into the transition function of M_1, so that M_1, at the start of each work period, can write it on a working track of the base colony. Then the work period will interpret it, applying it to the data found there, resulting in the simulation of M_2.

In this way, an infinite sequence of simulations arises, in order to withstand larger and larger but sparser and sparser bursts of faults.

Since the M_1 uses the universal interpreter, which in turns simulates the same program, it is natural to ask how machine M_1 simulates a given Turing machine G that does the actual useful computation? For this task, we set aside a separate track on each machine M_i, on which some arbitrary other Turing machine can be simulated. The higher the level of the machine M_k that performs this "side-simulation", the higher the reliability. Thus, only the simulations $M_k \to M_{k+1}$ are forced, without program (that is a hard-wired program): the side simulations can rely on written programs, since the firm structure in the hierarchy $M_1, M_2, \ldots$ will support them reliably.

2.3 From combinatorial to probabilistic noise

The construction we gave in the previous subsection was related to increasing bursts that are not frequent. In essence, that noise model is combi-

natorial. To deal with probabilistic noise combinatorially, we stratify the set of faulty times *Noise* as follows. For a series of parameters β_k, V_k, we first remove "isolated bursts" of type (β_1, V_1) of elements of this set. (The notion of "isolated bursts" of type (β, V) will be defined appropriately.) Then, we remove isolated bursts of type (β_2, V_2) from the remaining set, and so on. It will be shown that with the appropriate choice of parameters, with probability 1, eventually nothing is left over from the set *Noise*.

A composition of two reliable simulations is even more reliable. We will see that a sufficiently large hierarchy of such simulations resists probabilistic noise.

2.4 Difficulties

Let us spell-out some of the main problems that the paper deals with, and some general ways they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

Non-aligned colonies A large burst of M_1 can modify the order of entire colonies or create new ones with gaps between them.

To overcome this problem conceptually, we introduce the notion of a *generalized Turing machine* allowing for non-adjacent cells. Each such machine has a parameter B called the *cell body size*. The cell body size of a Turing machine in Section 1.2 would still remain 1.

No structure What to do when the head is in a middle of an empty area where no structure exists? To ensure reliable passage across such areas, we will try to keep everything filled with cells, even if these are not part of the main computation.

Clean areas Noise can create areas over which the predictability of the simulated machine is limited. In these areas the (on this level) invisible structure of the underlying simulation may be destroyed. These areas should not simply be considered blank, since blankness implies predictable behavior. We could call these areas "dirty", but we prefer to use only a positive terminology, and talk about the complement, namely *clean* intervals. There is a danger in using the negative terminology, since the transition properties will allow to make only be from cleanness, not from "dirtiness". The following example shows that when the head comes out of a clean area, it can be "sucked in", and its state can change.

Extending cleanness The definition of the generalized Turing machine stipulates a certain "magical" extension of clean intervals, and also a magical appearance of a clean "hole" around the head whenever it passes a certain amount of cumulative time in a small interval. (Of course, this property needs to be implemented in simulation, which is one of the main burdens of the actual construction.) Once an area is cleaned, it will be repopulated with new cells. Their content is not important, what matters is the restoration of predictability.

Rebuilding If local repair fails, a special rule will be invoked that reorganizes a larger part of the tape (of the size of a few colonies instead of only a few cells). This is the mechanism enabling the "magical" restoration on the next level.

Example 2.1 (Uncleanness) Consider two levels of simulation as outlined in Section 2.2: machine M_1 simulates M_2 which simulates M_3. The tape of M_1 is subdivided into colonies of size Q_1. The tape content of the current colony of level 1 represents not only the content of the currently observed tape cell of machine M_2, but also its state.

A burst on level 1 has size $O(1)$, while a burst on level 2 has size $O(Q_1)$. Suppose that such a burst happens at some time, while the head was performing a simulation in colony $C(x) = x + [0, Q_1)$, and just intending to leave it on the *left* for a long time. Let a burst $\mathbf{b}_1$ of level 2 change the right neighbor colony $C(x+Q_1)$ and the state represented by it completely, into the last stage of a work period with the head on the verge of leaving it on the *right*. After that, let the burst deposit the head onto $C(x)$ again, letting it finish its simulation and continue on the *left*, for a very long time.

Much later, the head returns to $C(x)$, and would still not visit $C(x+Q_1)$ by design. But when it is at the right edge of $C(x)$, a burst $\mathbf{b}_2$ of level 1 (which can happen essentially in every work period of level 1) moves the head over to the left edge of $C(x + Q_1)$. Here it will be captured, simulating the cell x of M_2 from a cell and machine state represented by colony $C(x + [0, Q_1)$ as set up by the big burst $\mathbf{b}_1$ earlier.

From the point of view of the simulated machine M_2 (which does not see bursts of level 1), the head came close to the unclean area created by burst $\mathbf{b}_1$, and then the state changed: the head started moving right, in a new state. ⌟

2.5 A shortcut solution

A fault-tolerant one-dimensional cellular automaton is constructed in [4]. If our Turing machine could just simulate such an automaton, it would become fault-tolerant. This can indeed almost be done provided that the size of the computation is known in advance. The cellular automaton can be made finite, and we could define a "kind of" Turing machine with a *circular tape* simulating it. But this solution requires input size-dependent hardware.

It seems difficult to define a fault-tolerant sweeping behavior on a regular Turing machine needed to simulate cellular automaton, without recreating an entire hierarchical construction – as we are doing here.

3 Notation

Most notational conventions given here are common; some other ones will also be useful.

Natural numbers and integers By $\mathbb{Z}$ we denote the set of integers.

$$\mathbb{Z}_{>0} = \{x : x \in \mathbb{Z},\ x > 0\},$$
$$\mathbb{Z}_{\geq 0} = \mathbb{N} = \{x : x \in \mathbb{Z},\ x \geq 0\}.$$

Intervals We use the standard notation for intervals:

$$[a, b] = \{x : a \leq x \leq b\}, \quad [a, b) = \{x : a \leq x < b\},$$
$$(a, b] = \{x : a < x \leq b\}, \quad (a, b) = \{x : a < x < b\}.$$

We will also write $[a, b)$ in place of $[a, b) \cap \mathbb{Z}$, whenever this leads to no confusion. Instead of $[x + a, x + b)$, sometimes we will write

$$x + [a, b)\,.$$

Ordered pairs Ordered pairs are also denoted by (a, b), but it will be clear from the context whether we are referring to an ordered pair or open interval.

Comparing the order of a number and an interval For a given number x and interval I, we write

$$x \geq I$$

if for every $y \in I$, $x \geq y$.

Distance The distance between two real numbers x and y is defined in a usual way:

$$d(x, y) = |x - y|.$$

The *distance of a point x from interval I* is

$$d(x, I) = \min_{y \in I} d(x, y).$$

Ball, neighborhood, ring, stripe A *ball of radius $r > 0$, centered at x* is

$$B(x, r) = \{y : d(x, y) \leq r\}.$$

An *r-neighborhood of interval I* is

$$\{x : d(x, I) \leq r\}.$$

An *r-ring* around interval I is

$$\{x : d(x, I) \leq r \text{ and } x \notin I\}.$$

An *r-stripe to the right of interval I* is

$$\{x : d(x, I) \leq r \text{ and } x \notin I \text{ and } x > I\}.$$

Logarithms Unless specified differently, the base of logarithms throughout this work is 2.

4 Specifying a Turing machine

Let us introduce the tools allowing to describe the reliable Turing machine.

4.1 Universal Turing machine

We will describe our construction in terms of universal Turing machines, operating on binary strings as inputs and outputs. We define universal Turing machines in a way that allows for rather general "programs".

Definition 4.1 (Standard pairing) For a (possibly empty) binary string $x = x(1)\cdots x(n)$ let us introduce the map

$$\langle x\rangle = 0^{|x|}1x,$$

Now we encode pairs, triples, and so on, of binary strings as follows:

$$\langle s,t\rangle = \langle s\rangle t,$$
$$\langle s,t,u\rangle = \langle\langle s,t\rangle,u\rangle,$$

and so on.

From now on, we will assume that our alphabets Σ, Γ are of the form $\Sigma = \{0,1\}^s$, $\Gamma = \{0,1\}^g$, that is our tape symbols and machine states are viewed as binary strings of a certain length. Also, if we write $\langle i,u\rangle$ where i is some number, it is understood that the number i is represented in a standard way by a binary string. ⌟

Definition 4.2 (Computation result, universal machine) Assume that a Turing machine M starting on binary x, at some time t arrives at the first time at some final state. Then we look at the longest (possibly empty) binary string to be found starting at position 0 on the tape, and call it the *computation result* $M(x)$. We will write

$$M(x,y) = M(\langle x,y\rangle), \quad M(x,y,z) = M(\langle x,y,z\rangle),$$

and so on.

A Turing machine U is called *universal* among Turing machines with binary inputs and outputs, if for every Turing machine M, there is a binary string p_M such that for all x we have $U(p_M,x) = M(x)$. (This equality also means that the computation denoted on the left-hand side reaches a final state if and only if the computation on the right-hand side does.) ⌟

Let us introduce a special kind of universal Turing machines, to be used in expressing the transition functions of other Turing machines. These are just the Turing machines for which the so-called s_{mn} theorem of recursion theory holds with $s(x,y) = \langle x,y\rangle$.

Definition 4.3 (Flexible universal Turing machine) A universal Turing machine will be called *flexible* if whenever p has the form $p = \langle p',p''\rangle$ then

$$U(p,x) = U(p',\langle p'',x\rangle).$$

Even if x has the form $x = \langle x', x'' \rangle$, this definition chooses $U(p', \langle p'', x \rangle)$ over $U(\langle p, x' \rangle, x'')$, that is starts with parsing the first argument (this process converges, since x is shorter than $\langle x, y \rangle$). ⌟

It is easy to see that there are flexible universal Turing machines. On input $\langle p, x \rangle$, a flexible machine first checks whether its "program" p has the form $p = \langle p', p'' \rangle$. If yes, then it applies p' to the pair $\langle p'', x \rangle$. (Otherwise it just applies p to x.)

Definition 4.4 (Transition program) Consider an arbitrary Turing machine M with state set Γ, alphabet Σ, and transition function τ. A binary string π will be called a *transition program* of M if whenever $\tau(a, q) = (a', q', j)$ we have

$$U(\pi, a, q) = \langle a', q', j \rangle.$$

We will also require that the computation induced by the program makes $O(|p| + |a| + |q|)$ left-right turns, over a length tape $O(|p| + |a| + |q|)$. ⌟

The transition program just provides a way to compute the (local) transition function of M by the universal machine, it does not organize the rest of the simulation.

Remark 4.5 In the construction of universal Turing machines provided by the textbooks (though not in the original one given by Turing), the program is generally a string encoding a table for the transition function τ of the simulated machine M. Other types of program are imaginable: some simple transition functions can have much simpler programs. However, our fixed machine is good enough (similarly to the optimal machine for Kolmogorov complexity). If some machine U' simulates M via a very simple program q, then

$$M(x) = U'(q, x) = U(p_{U'}, \langle q, x \rangle) = U(\langle p_{U'}, q \rangle, x),$$

so U simulates this computation via the program $\langle p_{U'}, q \rangle$. ⌟

4.2 Rule language

In what follows we will describe the generalized Turing machines M_k for all k. They are all similar, differing only in the parameter k; the most important activity of M_k is to simulate M_{k+1}. The description will be

uniform, except for the parameter k. We will denote therefore M_k simply by M, and M_{k+1} by M^*. Similarly we will denote the block size Q_k of the block code of the simulation simply by Q.

Instead of writing a huge table describing the transition function $\tau_k = \tau$, we present the transition function as a set of *rules*. It will be then possible to write one *interpreter* program that carries out these rules; that program can be written for some fixed flexible universal machine Univ.

Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then** *instruction* **else** *instruction*", where the condition is testing values of some fields of the state and the observed cell, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of the state or cell symbol, or a command to move the head. Rules can call other rules, but these calls will never form a cycle. Calling other rules is just a shorthand for nested conditions.

Even though rules are written like procedures of a program, they describe a single transition. When several consecutive statements are given, then they change different fields of the state or cell symbol, so they can be executed simultaneously.

Assignment of value x to a field y of the state or cell symbol will be denoted by $y \leftarrow x$. We will also use some conventions introduced by the C language: namely, $x \leftarrow x + 1$ and $x \leftarrow x - 1$ are abbreviated to $x{+}{+}$ and $x{-}{-}$ respectively.

Rules can also have parameters, like $\texttt{Swing}(a, b, u, v)$. Since each rule is called only a constant number of times in the whole program, the parametrized rule can be simply seen as a shorthand.

Mostly we will describe the rules using plain English, but it should always be clear that they are translatable into such rules.

For the machine M we are constructing, each state will be a tuple $q = (q_1, q_2, \ldots, q_k)$, where the individual elements of the tuple will be called *fields*, and will have symbolic names. For example, we will have fields *Addr* and *Drift*, and may write q_1 as $q.Addr$ or just *Addr*, q_2 as $q.Drift$ or *Drift*, and so on.

Similarly for tape symbols. In order to distinguish fields of tape symbols from fields of the state, we will always start the name of a field of the tape symbols by the letter c. We have seen already one example of this, the field *cDir* of tape symbols in the definition of a generalized Turing machine.

In what follows we describe some of the most important fields we will use; others will be introduced later.

A properly formatted configuration of M splits the tape into blocks of Q consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The colony that corresponds to the cell that the simulated machine is scanning is called the *base colony* (a precise definition will be based on the actual history of the work of M). Once the the direction of the simulated head movement, called the *drift*, is known, the union of the base colony with the target colony in the direction of the drift is called the *workspace* (this notion will need to be defined more carefully later).

There will be a field of the state called the *mode*:

$$Mode \in \{\text{Normal}, \text{Healing}, \text{Rebuilding}\}.$$

In the *normal* mode, the machine is engaged in the regular business of simulation. The *healing* mode tries to correct some local fault due to a couple of neighboring bursts, while the *rebuilding* mode attempts to restore the colony structure on the scale of a couple of colonies.

The content of each cell of the tape of M also has several fields. Some of these have names identical to fields of the state. In describing the transition rule of M we will write, for example, $q.Addr$ simply as $Addr$, and for the corresponding field of the observed cell symbol a we will write $a.cInfo$, or just $cInfo$. The array of values of the same field of the cells will be called a *track*. Thus, we will talk about the *cHold* track of the tape, corresponding to the *cHold* field of cells.

Each field of a cell has also a possible value $\emptyset$ whose approximate meaning is "undefined".

Some fields and parameters are important enough to introduce them right away. The

$$cInfo, cState$$

track of a colony of M contain the strings that encode the content of the simulated cell of M^* and its simulated state respectively.

$$cProg$$

track stores the program of M^*, in an appropriate form to be interpreted by the simulation. The field

$$cAddr$$

of the cell shows the position of the cell in its colony: it takes values in $[-Q, 2Q)$, since the addresses in a bridge (see later) will be continuations of those in the colony (which run from 0 to $Q-1$). There is a corresponding *Addr* field of the state.

The direction in $\{-1, 1\}$ in which the simulated head moves will be denoted by

$$Drift.$$

There is a corresponding field *cDrift*. The number of the last sweep of the work period will depend on the drift d, and will be denoted by

$$\text{Last}(d). \tag{4.1}$$

The colony along with the adjacent cells that continue its addresses will be called an *extended colony*. A colony can only be extended in the direction of the drift. The

$$Sw$$

field counts the sweeps that the head makes during the work period. There is a corresponding *cSw* field in the cell. In calculating parameters, we will make use of

$$V = \max(\text{Last}(-1), \text{Last}(1)). \tag{4.2}$$

Cells will be designated as belonging to a number of possible *kinds*, signaled by the field

$$cKind$$

with values

$$\text{Member}, \text{Target}, Vac, \text{Stem}.$$

Here is a description of the role of these cell kinds. Normally, cells will have the kind Member. During the simulation, however, the elements of the colony that is to become the next base colony, will be made to have the kind Target. If the neighbor colony is not adjacent, then the base colony will be extended to it by a *bridge* of up to $Q-1$ adjacent cells. Bridge cells are member cells with addresses outside $[0, Q)$. Though they have the kind

Member, they will frequently be treated differently from the other member cells.

The stem kind is sometimes convenient when some cells need to be created temporarily that do not participate in any known colony structure. We will also try to keep all areas between colonies filled with (not necessarily adjacent) stem cells. For example the computation may find that a colony does not properly encode a tape cell by the required error-correcting code. Then we want to "kill" the whole colony. This will happen by turning the kind of each of its cell to Stem.

During healing, some special fields of the state and cell are used, they will be subfields of the fields

$$\mathit{Heal} \tag{4.3}$$

and *cHeal* respectively. In particular, there will be *Heal.Sw* and *cHeal.Sw* fields. We will say that a cell is *marked for healing* if $\mathit{cHeal.Sw} \neq 0$. Similarly, during rebuilding we will work with subfields of the field *Rebd* and *cRebd*, and a cell will be called *marked for rebuilding* if $\mathit{cRebd.Sw} \neq 0$.

5 Exploiting structure in the noise

5.1 Sparsity

Let us introduce a technique connecting the combinatorial and probabilistic noise models.

Definition 5.1 (Centered rectangles, isolation) Let $\boldsymbol{r} = (r_1, r_2)$, $r_1, r_2 \geq 0$, be a two-dimensional nonnegative vector. An *rectangle* of radius $\boldsymbol{r}$ *centered* at $\boldsymbol{x}$ is

$$B(\boldsymbol{x}, \boldsymbol{r}) = \{\boldsymbol{y} : |y_i - x_i| \leq r_i, i = 1, 2\}. \tag{5.1}$$

Let $E \subseteq \mathbb{Z}^2$ be a two-dimensional set. A point $\boldsymbol{x}$ of E is $(\boldsymbol{r}, \boldsymbol{r}^*)$-*isolated* if

$$E \cap B(\boldsymbol{x}, \boldsymbol{r}^*) \subseteq B(\boldsymbol{x}, \boldsymbol{r}).$$

Set E is $(\boldsymbol{r}, \boldsymbol{r}^*)$-*sparse* if $D(E, \boldsymbol{r}, \boldsymbol{r}^*) = \emptyset$, that is it consists of $(\boldsymbol{r}, \boldsymbol{r}^*)$-isolated points. Let

$$D(E, \boldsymbol{r}, \boldsymbol{r}^*) = \{\boldsymbol{x} \in E : \boldsymbol{x} \text{ is not } (\boldsymbol{r}, \boldsymbol{r}^*)\text{-isolated from } E\}. \tag{5.2}$$

⌟

Definition 5.2 (Sparsity) Let

$$\beta \geq 9, \gamma \gg \beta \tag{5.3}$$

constants (we will choose γ sufficiently large as the proof requires), and let $0 < B_1 < B_2 < \cdots$, $T_1 < T_2 < \cdots$, be sequences of positive integers to be fixed later.

For a two-dimensional set E, let $E^{(1)} = E$. For $k > 1$ we define recursively:

$$E^{(k+1)} = D(E^{(k)}, \beta(B_k, T_k), \gamma(B_{k+1}, T_{k+1})). \tag{5.4}$$

Set $E^{(k)}$ is called the *k-th residue* of E. It is *k-sparse* if $E^{(k+1)} = \emptyset$. It is simply *sparse* if $\bigcap_k E^{(k)} = \emptyset$.

When $E = E^{(k)}$ and k is known then we will denote $E^{(k+1)}$ simply by E^*. ⌟

The following lemma connects the above defined sparsity notions to the requirement of small fault probability. It is formulated somewhat redundantly, for easier application.

Lemma 5.3 (Sparsity) *Let $Q_k = B_{k+1}/B_k$, $U_k = T_{k+1}/T_k$, and*

$$\lim_{k\to\infty} \frac{\log(U_k Q_k)}{1.5^k} = 0. \tag{5.5}$$

For sufficiently small ε, for every $k \geq 1$ the following holds. Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a random set with the property that each pair (p, t) belongs to E independently from the other ones with probability $\leq \varepsilon$.

Then for each point $\boldsymbol{x}$ and each k,

$$\mathbb{P}\{ B(\boldsymbol{x}, (B_k, T_k)) \cap E^{(k)} \neq \emptyset \} < 2\varepsilon \cdot 2^{-1.5^k}.$$

As a consequence, the set E is sparse with probability 1.

Proof. Let $k = 1$. Rectangle $B(\boldsymbol{x}, (B_1, T_1))$ is a single point, hence the probability of our event is $< \varepsilon$. Let us prove the inequality by induction, for $k + 1$.

Note that our event depends at most on the rectangle $B(\boldsymbol{x}, 3(B_k, T_k))$. Let

$$p_k = 2\varepsilon \cdot 2^{-1.5^k}.$$

Suppose $\boldsymbol{y} \in E^{(k)} \cap B(\boldsymbol{x}, \gamma(B_{k+1}, T_{k+1}))$. Then, according to the definition of $E^{(k)}$, there is a point

$$\boldsymbol{z} \in B(\boldsymbol{y}, \gamma(B_{k+1}, T_{k+1})) \cap E^{(k)} \setminus B(\boldsymbol{y}, \beta(B_k, T_k)). \tag{5.6}$$

Consider a standard partition of the (two-dimensional) space-time into rectangles $K_p = \boldsymbol{c}_p + [-B_k, B_k) \times [-T_k, T_k)$ with centers $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots$. The rectangles K_i, K_j containing $\boldsymbol{y}$ and $\boldsymbol{z}$ respectively intersect $B(\boldsymbol{x}, 2\gamma(B_{k+1}, T_{k+1}))$. The triple-size rectangles $K_i' = c_i + [-3B_k, 3B_k) \times [-3T_k, 3T_k)$ and K_j' are disjoint, since (5.3) and (5.6) imply $|y_1 - z_1| > \beta B_k$ and $|y_2 - z_2| > \beta T_k$.

The set $E^{(k)}$ must intersect two rectangles K_i, K_j of size $2(B_k, T_k)$ separated by at least $4(B_k, T_k)$, of the big rectangle $B(\boldsymbol{x}, 2\gamma(B_{k+1}, T_{k+1}))$.

By the inductive hypothesis, the event $\mathcal{F}_i$ that K_i intersects E_k has probability bound p_k. It is independent of the event $\mathcal{F}_j$, since these events depend only on the triple size disjoint rectangles K_i' and K_j'.

The probability that both of these events hold is at most p_k^2. The number of possible rectangles K_p intersecting $B(\boldsymbol{x}, 2\gamma(B_{k+1}, T_{k+1}))$ is at most $C_k := ((2\gamma^2 U_k Q_k) + 2)^2$, so the number of possible pairs of rectangles is at most $C_k^2/2$, bounding the probability of our event by

$$\begin{aligned} C_k^2 p_k^2/2 &= 2C_k^2 \varepsilon^2 2^{-1.5^{k+1}} \cdot 2^{-0.5 \cdot 1.5^k} \\ &= 2\varepsilon 2^{-1.5^{k+1}} \cdot \varepsilon C_k^2 2^{-0.5 \cdot 1.5k}. \end{aligned}$$

Since $\lim_k \frac{\log (U_k Q_k)}{1.5^k} = 0$, the last factor is ≤ 1 for sufficiently small ε. □

5.2 Error-correcting code

Let us add error-correcting features to block codes introduced in Definition 1.2.

Definition 5.4 (Error-correcting code) A block code is (β, t)-*burst-error-correcting*, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\psi^*(y) = x$ whenever y differs from $\psi_*(x)$ in at most t intervals of size $\leq \beta$.

For such a code, we will call a word $y \in \Sigma_1^Q$ is r-*compliant* if it differs from a codeword of the code by at most r intervals of size $\leq \beta$. ⌟

Example 5.5 (Repetition code) Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. Let $\psi^*(y)$ be obtained as follows. If $y = y(1) \ldots y(Q)$, then $x = \psi^*(y)$ is defined as follows: $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y+2Q/3)$. For all $\beta \leq Q/3$, this is a $(\beta, 1)$-burst-error-correcting code.

If we repeat 5 times instead of 3, we get a $(\beta, 2)$-burst-error-correcting code. Let us note that there are much more efficient such codes than just repetition. ⌟

Consider a Turing machine $(\Gamma, \Sigma, \tau, q_{\text{start}}, F)$ (actually a generalized one to be defined later) simulating some Turing machine $(\Gamma^*, \Sigma^*, \tau^*, q^*_{\text{start}}, F^*)$. We will assume that $\Gamma^* \cup \{\emptyset\}$, and the alphabet Σ^* are subsets of the set of binary strings $\{0,1\}^\ell$ for some $\ell < Q$ (we can always ignore some states or tape symbols, if we want).

Definition 5.6 (Interior) Let *PadLen* be a parameter to be defined below, in (7.2). We will call the interval of cells of a colony with addresses in $[PadLen, Q - PadLen)$ the *interior* of a colony. ⌟

We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries. So let (υ_*, υ^*) be a $(\beta, 2)$-burst-error-correcting block code

$$\upsilon_* : \{0,1\}^\ell \cup \{\emptyset\} \to \{0,1\}^{(Q-2\cdot PadLen)B}.$$

We could use, for example, the repetition code of Example 5.5. Other codes are also appropriate, but we require that they have some fixed programs p_{encode}, p_{decode} on the universal machine Univ, in the following sense:

$$\upsilon_*(x) = \text{Univ}(p_{\text{encode}}, x), \quad \upsilon^*(y) = \text{Univ}(p_{\text{decode}}, y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as the repetition code certainly does).

Let us now define the block code (ψ_*, ψ^*) used in the definition of the configuration code (φ_*, φ^*) as outlined in Section 6.3. We define

$$\psi_*(a) = 0^{PadLen} \upsilon_*(a) 0^{PadLen}. \tag{5.7}$$

It will be easy to compute the configuration code from ψ_*, once we know what fields there are which ones need initialization.

The decoded value $\psi^*(x)$ is obtained by first removing *PadLen* symbols from both ends of x to get x', and then computing $\upsilon^*(x')$.

6 The model

Recall the definition of sparsity in Section 5.1: there will be a sequence $0 < B_1 < B_2 < \cdots$ of "scales" in space and a sequence $T_1 < T_2 < \cdots$ of

scales in time, and a constant β. We will define a sequence of simulations $M_1 \to M_2 \to \cdots$ where each M_k is a machine simulating one on a higher level. For simplicity, we will use the notation $M = M_k$, $M^* = M_{k+1}$, and similarly for the other parameters, for example B, T, Q, U. As already indicated in Section 2.2, these machines will generalize ordinary Turing machines, with a number of new features.

6.1 Generalized Turing machine

Standard Turing machines do not have operations like "creation" or "killing" of cells, nor do they allow for cells to be non-adjacent. We introduce here a *generalized Turing machine*. It depends on an integer $B \geq 1$ that denotes the cell body size, and an upper bound T on the transition time, as well as a *level number* λ, whose meaning will be explained in the definition of configurations. These parameters are convenient since they provide the illusion that the different Turing machines in the hierarchy of simulations all operate on the same linear space. Even if the notions of cells, alphabet and state are different for each machine of the hierarchy, at least the notion of a *location on the tape* is the same.

Definition 6.1 (Generalized Turing machine) A *generalized Turing machine* M is defined by a tuple

$$(\Gamma, \Sigma, \tau, \mathit{NonAdj}, \mathit{cDir}, q_{\text{start}}, F, B, T, \lambda), \tag{6.1}$$

where Γ and Σ are finite sets called the *set of states* and the *alphabet* respectively,

$$\tau : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{-1, 1\},$$

is the *transition function*. The function ("field") $\mathit{NonAdj} : \Gamma \to \{\mathsf{true}, \mathsf{false}\}$ of the state will show whether the last move was from a non-adjacent cell. The function ("field") $\mathit{cDir} : \Sigma \to \{-1, 1\}$ of the cell content needs to always point towards the head, so the transition function τ is required to have the property that if $(a', q', j) = \tau(a, q)$ then $a'.\mathit{cDir} = j$.

The role of starting state q_{start} and final states in F is as before. The integer $B \geq 1$ is called the *cell body size*, and the real number T is a bound on the transition time. The level λ will play a role in the definition of trajectories.

Among the elements of the tape alphabet Σ, we distinguish the elements $0, 1, Bad, Vac$. The role of the symbols Bad and Vac will be clarified below.

⌟

Definition 6.2 (Configuration) Consider a generalized Turing machine (6.1). A *configuration* is a tuple

$$(q, L, A, h, \tilde{h},),$$

where $q \in \Gamma$, $L : \mathbb{Z} \to [0, \lambda] \cap \mathbb{N}$, $A : \mathbb{Z} \to \Sigma$, $h, \tilde{h} \in \mathbb{Z}$. The arrays L, A make up the tape configuration. The value $0 \leq L(p) \leq \lambda$ shows the *level* of a position p on the tape. A point is *clean* if $L(p) = \lambda$, the maximum possible level. Informally, even in unclean points p, the value $L(p)$ measures the level of "organization" of a certain neighborhood of p. The cell state $A(p)$ will have importance only in case $L(p) = \lambda$. A set of points is *clean* if it consists of clean points.

We say that there is a *cell* at a position $p \in \mathbb{Z}$ if the interval $p + [0, B)$ is clean and $A(p) \neq Vac$. In this case, we call the interval $p + [0, B)$ the *body* of this cell. Cells must be at distance $\geq B$ from each other, that is their bodies must not intersect. They are called *adjacent* if the distance is exactly B.

For all cells p, the value $A(p).cDir$ is required to point towards the head position h, that is

$$A(p).cDir = \text{sign}(h - p).$$

If the head is within a clean interval, at a distance

Whenever the interval $h + [3B, 3B)$ is clean there must be a cell at some position $\tilde{h}$ within this interval called the *current cell*, with a body within $2B$ from h.

The array A is Vac everywhere but in finitely many positions.

Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine M.

⌟

All the above definitions can clearly be localized to define a configuration *over a space interval* I, where it is always understood that $h \in I$, that is I contains the head.

Definition 6.3 (Local configuration, replacement) A *local configuration on* a (finite or infinite) interval I is given by values assigned to the cells of I, along with the following information: whether the head is to the left of, to the right of or inside I, and if it is inside, on which cell, and what is the state.

If I' is a subinterval of I, then a local configuration ξ on I clearly gives rise to a local configuration $\xi(I')$ on I' as well, called its *subconfiguration*: If the head of ξ was in I and it was for example to the left of I', then now $\xi(I')$ just says that it is to the left, without specifying position and state.

Let ξ be a configuration and $\zeta(I)$ a local configuration that contains the head if and only if $\xi(I)$ contains the head. Then the configuration $\xi|\zeta(I)$ is obtained by replacing ξ with ζ over the interval I, further if ξ contains the head then also replacing $\xi.\mathrm{pos}$ with $\zeta.\mathrm{pos}$ and $\xi.\mathrm{state}$ with $\zeta.\mathrm{state}$. ⌟

It is natural to name a sequence of configurations that is conceivable as a computation (faulty or not) of a Turing machine as "history". The histories that obey the transition function then could be called "trajectories". In what follows we will stretch this notion to encompass also some limited violations of the transition function.

In connection with any underlying Turing machine with a given starting configuration, we will denote by

$$Noise \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0} \tag{6.2}$$

the set of space-time points (p, t), such that a fault occurs at time t when the head is at position p.

Definition 6.4 (History) Let us be given a generalized Turing machine (6.1). Consider a sequence $\eta = (\eta(0), \eta(1), \dots)$ of configurations with $\eta(t) = (q(t), L(t), A(t), h(t), \tilde{h}(t))$, along with a noise set $Noise$. The *switching times* of this sequence are the times when one of the following can change: the state, the position $\tilde{h}(t)$ of the current cell, the level and the state of the current cell. (The level can also change at non-switching times.) The interval between two consecutive switching times is the *dwell period*. The pair

$$(\eta, Noise)$$

will be called a *history* of machine M if the following conditions hold.

- We have $|h(t) - h(t')| \le |t' - t|$.
- In two consecutive configurations, the level $L(p,t)$ and content $A(p,t)$ of the positions p not in $h(t) + [-B, B)$, remains the same: for example $A(n, t+1) = A(n,t)$ for all $n \notin h(t)+[-B, B)$. ⟨⟨P: Maybe for the level we will use $[-2B, 2B)$.⟩⟩
- At each noise-free switching time the head is on the new current cell, that is $\tilde{h}(t) = h(t)$. In particular, when at a switching time a current cell becomes *Vac*, the head must already be on another (current) cell.
- The length of any noise-free dwell period in which the head is staying on clean positions is at most T.

Let

$$\text{Histories}_M$$

denote the set of all possible histories of M.

We say that a cell *dies* in a history if it becomes *Vac*.

It is clear that all the above definition can be *localized* to define a history *over a space-time rectangle* $I \times J$, where it is always understood that $h \in I$ for all times $t \in J$, that is I contains the head throughout the time interval considered.

⌟

The transition function τ of a generalized Turing machine imposes constraints on histories: those histories obeying the constraints will be called trajectories.

Definition 6.5 Suppose that the machine is in a state q, with current cell x, with cell content a, let $(a', q', j) = \tau(a, q)$. We say that the new content of cell x (including when it dies), the direction of the new position y from x, and the new state are *directed by the transition function* if the following holds. The new content of x is a', and the direction of y from x is j. In the new state q we have $q.NonAdj = \mathsf{false}$ if $j = 0$ or the new current cell is adjacent, and true otherwise. ⌟

Definition 6.6 (Trajectory) A history $(\eta, Noise)$ of a generalized Turing machine (6.1) with $\eta(t) = (q(t), L(T), A(t), h(t), \tilde{h}(t))$ is called a *trajectory* of M if the following conditions hold, during any noise-free time interval. Denote

$$\begin{aligned}(a, q, p) &= (A(\tilde{h}(t), t), q(t), \tilde{h}(t)),\\ (a', q', j) &= \tau(a, q).\end{aligned}$$

Transition Function Suppose that there is a switch, and the shortest interval containing the body of the current cell x and the new cell y is to at least a distance $c_{\text{c-depth-1}}B$ inside a clean interval, where

$$c_{\text{c-depth-1}} = 0.5. \tag{6.3}$$

Then the new state, the cell content of x (including when it dies), and the direction of y from x are directed by the transition function. If y did not exist before then it is adjacent to x. Nothing else changes on the tape.

Further the length of the dwell period is bounded by T.

Spill Bound Recall the constant γ introduced in Definition 5.2. Consider a noise-free space-time rectangle $[a, b) \times [u, v)$ in which $[a, b)$ is clean at time u. Let

$$s = (b - a)(\beta/\gamma).$$

Then $[a + s, b - s)$ is clean at time v.

Attack Cleaning For constant

$$c_{\text{c-depth-2}} = 3, \tag{6.4}$$

and current cell x, suppose that the interval $[x - c_{\text{c-depth-2}}B, x + B)$ is clean. Suppose further that the transition function directs the head right. Then by the time the head comes back to $x - c_{\text{c-depth-2}}B$, there is a cell $y \in [x + B, x + 2B)$ such that the larger interval $[x - c_{\text{c-depth-2}}B, y + B)$ is also clean.

A similar property is required when "left" and "right" are interchanged.

Dwell Cleaning For constants

$$c_{\text{clean-t}} = 9, \quad c_{\text{clean-s}} = c_{\text{c-depth-2}} + 1, \tag{6.5}$$

suppose that during a noise-free time interval $[u, v]$, the head spends time $c_{\text{clean-t}}T$ in a certain interval of size B without leaving it. Then at time v, the interval $h + [c_{\text{clean-s}}B, c_{\text{clean-s}}B)$ around its position h is clean.

Pass Cleaning Consider a noise-free time interval J, and the space interval I of all positions of the head during J. Let $p(t)$ be the head position at time t. Suppose $L(p(t)) < \lambda$ for all t in $J/$ For appropriate constants

$$0 < c_{\text{decr}} < c_{\text{incr}},$$

the scorescore(I) increases by at least

$$c_{\text{incr}}(1 + 2^{-\lambda})|I| - c_{\text{decr}}B.$$

The subtracted term needs to be present only if the head starts but does not end on a clean position.

⌟

The above definition can also clearly be localized to some space-time rectangle just as the definition of history was.

The Attack property says essentially that if the head moves out on the right end of a clean interval then next time it comes in, it must extend the right end of the interval by at least B (while temporarily possibly withdrawing it by $c_{\text{spill}}B$). The Dwell Cleaning property says essentially that if the head spends a certain amount of time in a small area J then it either must clean out an interval of certain size around itself or move into such an interval that is already clean. The Pass Cleaning property is similar, but it does not require spending a certain amount of time in J, rather it requires the head to pass across J a certain number of times. It will be used in the proof of the Spill Bound property of simulated trajectories.

The transition function we are going to define will keep the head in a zigzag motion; these properties will enable it then to clean out large areas.

6.2 Simulation

Until this moment, we used the term "simulation" informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code $\varphi = (\varphi_*, \varphi^*)$. As a matter of fact, the *decoding* part of the code is the more important. Indeed, we want to say that machine M_1 simulates machine M_2 via simulation ϕ if whenever (η, Noise) is a trajectory of M_1 then $(\eta^*, \mathit{Noise}^*)$, defined by $\eta^*(\cdot, t) = \phi^*(\eta(\cdot, t))$, is a trajectory of M_1. Here, Noise^* is computed by an appropriate mapping.

We will make, however, two refinements. First, it is not important to require this for arbitrary trajectories η of M_1: it is sufficient to consider those η for which the initial configuration $\eta(\cdot, 0)$ has been obtained by encoding, that is it has the form $\eta(\cdot, 0) = \varphi_*(\xi)$. So the encoding function comes in, after all.

But there is a more complex refinement. When a colony is in transition between encoding one simulated value to encoding another one, there may be times when the value represented by it before the transition is already not decodable from it, and the value after the transition is not yet decodable from it. For this reason, it is useful to define a notion of simulation decoding Φ^* as a mapping between *histories*, not just configurations. We will not abuse this notion, but it will allow us a certain amount of looking back: the map Φ^* can use the configuration at the beginning of the work period for decoding.

We will also give another function to the mapping Φ^*. A history was defined above in Definition 6.4 as a pair (η, Noise), so Φ^* also computes a new noise set: $\Phi^*(\eta, \mathit{Noise}) = (\eta^*, \mathit{Noise}^*)$. The meaning of this new noise set will be, just as in Definition 5.2 of sparsity, that Noise^* will be obtained by deleting some small isolated parts of Noise that the error-correcting simulation can deal with.

Definition 6.7 (Simulation) Let M_1, M_2 be two generalized Turing machines, and let

$$\varphi_* : \mathrm{Configs}_{M_2} \to \mathrm{Configs}_{M_1}$$

be a mapping from configurations of M_2 to those of M_1, such that it maps starting configurations into starting configurations. We will call such a map a *configuration encoding*. Let

$$\Phi^* : \mathrm{Histories}_{M_1} \to \mathrm{Histories}_{M_2}$$

be a mapping. The pair (φ_*, Φ^*) is called a *simulation* (of M_2 by M_1) if for every trajectory (η, Noise) with initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$, the history $(\eta^*, \mathit{Noise}^*) = \Phi^*(\eta, \mathit{Noise})$ is a trajectory of machine M_2.

We say that M_1 *simulates* M_2 if there is a simulation (φ_*, Φ^*) of M_2 by M_1. ⌟

6.3 Hierarchical codes

Recall the notion of a code in Definition 1.2.

Definition 6.8 (Code on configurations) Consider two generalized Turing machines M_1, M_2 with the corresponding state spaces, alphabets and transition functions, and an integer $Q \geq 1$. We require

$$B_2 = QB_1. \tag{6.6}$$

Assume that a block code

$$\psi_* : \Sigma_2 \times (\Gamma_2 \cup \{\emptyset\}) \to \Sigma_1^Q$$

is given, with an appropriate decoding function, ψ^*. With $(a, q) \in \Sigma_2 \times (\Gamma_2 \cup \{\emptyset\})$, symbol a is interpreted the content of some tape square. The value q is the state of M_2 provided the head is observing this square, and $\emptyset$ if it is not.

Block code (ψ_*, ψ^*) gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}, \quad \varphi^* : \text{Configs}_{M_1} \to \text{Configs}_{M_2}$$

that encodes configurations of M_2 into configurations of M_1.

Let ξ be a configuration of M_2. We set $\varphi_*(\xi).\text{pos} = \xi.\text{pos}$, $\varphi_*.\text{state} =$ the starting state of M_1,

$$\varphi_*(\xi).\text{tape}[iB_2, \ldots, (i+1)B_2 - B_1] = \psi_*(\xi.\text{tape}[i], s)$$

where $s = \xi.\text{state}$ if $i = \xi.\text{pos}$, and $\emptyset$ otherwise, with a slight *modification*: $\varphi_*(\xi).\text{tape}[i].cDir$, is set to point towards the head $\xi.\text{pos}$.

A configuration ξ is called a *code configuration* if it has the form $\xi = \varphi_*(\zeta)$. ⌟

Definition 6.9 (Hierarchical code) For $k \geq 1$, let Σ_k be an alphabet, Γ_k be a set of states of a generalized Turing machine M_k. Let $Q_k > 0$ be an integer colony size, let φ_k be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \times (\Gamma_{k+1} \cup \{\emptyset\}) \to \Sigma_k^{Q_k}$$

as in Definition 6.8. The sequence of triples $(\Sigma_k, \Gamma_k, \varphi_k)$, $(k \geq 1)$, is called a *hierarchical code*. For the given hierarchical code, the configuration ξ^1 of M_1 is called a *hierarchical code configuration* if a sequence of configurations $\xi^2, \xi^3, \ldots$ of $M_2, M_3, \ldots$ exists with

$$\xi^k = \varphi_{*k}(\xi^{k+1})$$

for all k. (Of course, then whole sequence is determined by ξ^1.)

Let M_1, M_2, ..., be a sequence of generalized Turing machines, let φ_1, φ_2 , ... be a hierarchical code for this sequence, let ξ^1 be a hierarchical code

configuration for it, where ξ^k is an initial configuration of M_k for each k. Let further be a sequence of mappings Φ_1, Φ_2, ... be given such that for each k, the pair (φ_{k*}, Φ_k^*), is a simulation of M_{k+1} by M_k. Such an object is called a *tower*. ⌟

The main task of the work will be the definition of a tower, since the simulation property is highly nontrivial.

7 Simulation structure

In what follows we will describe the program of the reliable Turing machine (more precisely, a simulation of each M_{k+1} by M_k as defined above). Most of the time, we will just refer to M_k as M and to M_{k+1} as M^*. Cells will be grouped into colonies, where $Q = B^*/B$ is the colony size. The behavior of the head on each colony simulates the head of M^* on the corresponding cell of M^*. The process takes a number of steps, constituting a *work period*.

Definition 7.1 The parameter $U = T^*/T$ is defined to be twice the maximum number of steps that any simulation work period can take. ⌟

Machine M will be able to perform the simulation even if the noise in which it operates is $(\beta(B,T),(B^*,T^*))$-sparse. By the above definitions, sparsity means that noise comes in *bursts* that are confined to rectangles of size $\beta(B,T)$ (affecting at most β consecutive tape cells), and are separated from each other in time in such a way that there is at most one burst in any two neighboring work period.

Along the way, we will try to give as much motivation as possible, including some definitions (like health) that will help in the proof later. There are some difficulties faced by our desire for a structured presentation: We cannot analyze unconditionally the error-correcting performance of a part of the program without seeing first the whole. Indeed, the noise can bring the machine into some state corresponding to an arbitrary part of the program.

We mentioned modes in Section 4.2. Ordinary simulation proceeds in the normal mode. To see whether the basic structure supporting this process is broken somewhere, each step will check whether the present state is *coordinated* with the currently observed cell symbol (see Definition 8.4). If not then the rule

`Heal`

takes the state into the healing mode. We will also say that *alarm* will be called. On the other hand, the state enters into rebuilding mode on some indications that healing fails. The crudest outline of the main rule of machine M is given in Rule 7.1; the `Compute` and `Transfer` rules will be outlined below.

Rule 7.1: Main rule
if the mode is normal **then**
if not Coordinated **then** `Heal`
else if $1 \leq Sw <$ TransferSw(1) **then** `Compute`
else if TransferSw(1) $\leq Sw <$ Last **then** `Transfer`
else if Last $\leq Sw$ **then** move the head to the new base.

7.1 Head movement

The global structure of a work period is this:

Computation phase The new simulated state and direction (called the drift) is computed. Then the "meaningfulness" of the result is checked. During this phase, the head sweeps, roughly, back-and-forth between the ends of the base colony.

Transfer phase The head moves into the neighbor colony in the simulated head direction, and transfers the simulated state to there. If the drift is, say, to the right, then the head sweeps, roughly, between the left end of the source colony and the right end of the target colony. There may be an area between the two colonies to bridge over.

The timing is controlled by a field Sw of the state and a field cSw of the cell state. We can read off the direction of the sweep s using the formula

$$\mathrm{dir}(s) = (-1)^{s+1}. \tag{7.1}$$

Some issues complicate the head movement, even in the absence of faults.

Zigging

A burst could turn the head back in the middle of its sweep. To detect such an event promtply, the sweeping will be complicated by a *zigzag* movement.

On its backward zig, the head can check whether the structure of last few cells is correct. Zigging will use certain parameters F, Z that we set to

$$\begin{aligned} F &= 7, \\ Z &= 14F\beta, \\ f_{\text{zig}} = 1, &\quad f_{\text{heal}} = 2, \quad f_{\text{rebuild}} = 3. \end{aligned}$$

The choice of the parameter F will be motivated by the discussion of feathering in Section 7.1. In its movement in direction $\delta \in \{-1, 1\}$ the head, after every forward sweeping move, will look for the first place ahead where it is allowed to turn back (see Section 7.1). As we will see this will be within F steps. It will move backwards $> Z$ steps to the first position with $Addr \equiv f_{\text{zig}} \pmod F$. We call *front* the farthest position of a cell that already was reached before the backward zig. The fields,

$$ZigDir \in \{-1, 1\}, ZigDepth \in [0, 2Z)$$

control the process. At the front, we have $ZigDepth = 0$ and $ZigDir = 1$. When the front reaches an appropriate turning point for zigging (see below), then we set $ZigDir \leftarrow -1$, and the head will be moving backwards from the front, *descending* into the zig, and increasing $ZigDepth$. When $ZigDepth$ reaches Z (or an allowed turning point with $ZigDepth > Z$, see below), we set $ZigDir \leftarrow 1$, the head turns towards the front and starts *ascending* from the zig, while decreasing $ZigDepth$.

Normally the head should reach the front exactly when $ZigDepth$ reaches 0. However, if it reaches the front earlier, then we just set $ZigDepth \leftarrow 0$. If it does not reach the front by this time, then the head will just keep marching forward, keeping $ZigDepth = 0$ until the front is reached. ⟨⟨P: Do we still have an example now that zigging begins only well inside?⟩⟩

Feathering

A somewhat arcane threat is responsible for another complication in head movement.

Example 7.2 Let $C(x)$ denote the colony with starting point x. Consider the following scenario.

1. At some time, at a last turn at the end of a right sweep on the right end of colony $C(x)$, at the bottom of the zig, say in position $y = x+(Q+Z)B$

if the zig goes outside the colony, a burst creates some dirt. Then the simulation dictates the head to leave $C(x)$ on the left.

2. Much later, on the first sweep of a return to $C(x)$, this dirt traps the head, and lets it start a right sweep to the right of y. When the head returns on a zig to y, a new burst corrects everything on the left of y, including y, but leaves the start of the new (false) right sweep on the right of y – even though the simulation does not dictate any move to the colony $C(x + QB)$.
3. Much later, the pair of events 1-2 happens again and again, allowing the started false sweep on the right of y to continue. After Z repetitions of this, the zig does not return already.

This way, a number of cleverly placed distant bursts can trigger an effect on the next level.

⌟

The sequence of Example 7.2 would not occur if our cellular automata had the following property:

Definition 7.3 (Feathering) A Turing machine execution is said to have the *feathering* property if the following holds. If the head turned back at a position x at some time, the next time the head arrives at position x it cannot turn back from it. ⌟

(The name "feathering" refers to the picture of the path of the head in a space-time diagram.) The property can be enforced by the following simple device: let the cell state have a field called

$$cCanTurn \in \{\mathsf{false}, \mathsf{true}\}.$$

We will allow the head to turn *only* in a cell with $cCanTurn = \mathsf{true}$. Whenever a head turns at a cell, it sets this value to 0; if it passes the cell, the value is reset to 1. For example, if the head is moving right and decides to turn left then it can do that at the first cell with $cCanTurn = \mathsf{true}$. Of course, a computation must then be reorganized in a way that this device does not prevent achieving its objectives. The following example suggests that such reorganization need not be too costly.

Example 7.4 (Feathering) Suppose that, arriving from the left at position 1, the head decides to turn left again. In repeated instances, it can then

turn back at the following sequence of positions:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 6, \ldots$$

⌟

This example shows that the feathering constraint does not divert the head too much. If in the original execution the head turned back t consecutive times to the left from position 0, then now it will turn back from somewhere in a zone of size $O(\log t)$ to the right of 0 in each of these times. It also shows that the exact turning point at the end of each sweep can be computed from the sweep number itself.

The simulated Turing machine will also have the feathering property, therefore it will not happen that the simulation turns back repeatedly from a colony. Note that it may still happen that at a point where it decided to turn, the head encounters a long run of cells with $cCanTurn = \mathsf{false}$. But this indicates the absence of a clean interval comparable with the size of this run – and the slide over such a run can be attributed to this.

Let us argue, informally, that one does not have to consider more than 3 islands in any area of size QB. We assume that the simulated program obeys the feathering condition: so if, say, the head comes from the left neighbor colony then at the end of the work period can turn back to the left only if the previous time it visited the colony, it has passed it over from right to left.

Example 7.5 (3 islands) Suppose that the head has arrived at C from the left, performs a work period and then passes to the right. In this case, if no new noise burst occurs then we expect that all islands found in C will be eliminated. On the other hand, a new island I_1 can be deposited. We can also assume that there is no island on the left of I_1 within distance QB, since the noise burst causing it would have been too close to the noise burst causing I_1.

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island I_1 will be eliminated, but a new one may be deposited. But what if the head turns back left at the end of the work period? If I_1 is close to the left end of C, then due to the feathering property, the head may never reach it sufficiently to eliminate it; moreover, it may add a new island I_2 on the right of I_1. We can also assume, similarly to the above, that there is no island on the right of I_2 within distance QB.

When the head returns a third time (possibly much later), from the right, it will have to leave on the left. The islands I_1, I_2 will be eliminated as they are passed over, but a possible new island I_3, created by a new burst (before, after or during the elimination), may remain. We can also assume, similarly to the above, that there is no island on the right of I_3 within distance QB. ⌟

Though the *cCanTurn* field assures feathering, we must make sure by organization that it is not preventing the turns that are needed. For this reason, some further regulations will be introduced. We will make use of a parameter E introduced in (11.2) in connection with the healing procedure. It is a multiple of F.

Zigging in normal mode Above, we made sure that the points for turning from left to right during simulation introduced at the bottom of a zig when sweeping right have $Addr \equiv f_{\text{zig}} \pmod F$. This makes sure that during left sweep, the head will always find a place within F cells with $cCanTurn = \mathsf{true}$.

Sweeping in normal mode Recall from (4.2) that the colony work period consists of at most V sweeps; we leave an area of

$$PadLen = E\lceil \log V \rceil + Z \tag{7.2}$$

cells on both ends of the colony free of simulation information, just assigned to feathering. Suppose that the head is to be turned back at "the right end". Then at a distance *PadLen* from the right end, it starts looking for a place to turn; it will actually turn back at the first position coming after this with $Addr \equiv f_{\text{end}} \pmod E$ and $cCanTurn = \mathsf{true}$. This will still leave enough time to find a turning point with $Addr < Q$, as remarked after Example 7.4.

Note that when the head changes direction at the end of a sweep, it has $Addr \equiv f_{\text{end}} \pmod E$; in other words, the possible turning points at the colony ends are separated from each other by the larger distance E which is a multiple of F. The rationale for this will become clear only when analyzing the boundary of a large clean and a large dirty area.

End turn of healing or rebuilding The healing and the rebuilding procedures need a turning point at the end of their ranges. In case the cell in question has an address (is not a stem cell) then it will need to have $Addr \equiv \pm f_{\text{heal}} \pmod F$ for healing turns and $\equiv \pm f_{\text{rebuild}}$ for rebuilding turns.

Zigging while healing Rebuilding mode will have its own internal addresses: as with healing (see later), to the left of the rebuilding front, the addresses are counted from the left end, and to the right of the head from the right end. When sweeping right then the bottom of the left zig will again be at a point with left address $\equiv f_{\text{zig}} \pmod F$.

We introduce a convenient notation.

Notation 7.6 Suppose that within a procedure we want the head to move n cells, and turn. It may not be able to turn then, but only at a point with $cCanTurn = \mathsf{true}$ and also having for example $cAddr \equiv \pm f_{\text{heal}}$ during healing. For simplicity, we will say that the head moves n^+ cells and turns. Thus n^+ is greater than n, but by not much: only an amount $O(\log \beta)$ for healing, and $O(\log Q)$ for simulation and rebuilding. ⌟

7.2 Computation phase

As shown in Rule 7.1 describing a top-down view of the simulation, the first phase of the simulation computes new values for the state of the simulated machine M^* represented on track *cState*, the direction of the move of the head of M^* (represented in the *cDrift* field of each cell of the colony of M), and the simulated cell state of M^* represented on the track *cInfo*. During this rule, the head sweeps the base colony.

Recall Definition 5.4. The rule `Compute` will rely on a certain fixed $(\beta, 3)$ burst-error-correcting code, moreover it expects that each of the words found on the *cState* and *cInfo* tracks is 2-*compliant*.

The rule `ComplianceCheck` checks whether a word is 2-compliant.

The rule `Compute` essentially repeats 3 times the following *stages*: decoding, applying the transition, encoding. Then it calls `ComplianceCheck`; if the latter fails it will mark the colony for rebuilding.

In more detail:

1. For every $j = 1, \ldots, 3$, if $Addr \in \{0, \ldots, Q-1\}$ do

    a) Calling by g the string found on the *cState* track of the interior of the base colony, decode it into string $\tilde{g} = \upsilon^*(g)$ (this should be the current state of the simulated machine), and store it on some auxiliary track in the base colony. Do this by simulating the universal machine on the *cProg* track, $\tilde{g} = \text{Univ}(p_{\text{decode}}, g)$.

Proceed similarly with the string a found on the *cInfo* track of the base colony, in the address interval J, to get $\tilde{a} = \upsilon^*(a)$ (this should be the observed tape symbol of the simulated machine).

Perform all this computation in a way using information from outside the address interval J, nor on any part of the state brought back into this interval other than the address, sweep and zigging fields.

b) Compute the value $(a', g', d) = \tau^*(\tilde{a}, \tilde{g})$. Since the neither the table nor any program of the transition function τ^* is written explicitly anywhere, this "self-simulation" step needs some elaboration, see Section 7.3.

c) Write the encoded new state $\upsilon_*(g')$ onto the $cHold[j].State$ track of the interior of the base colony. Similarly, write the encoded new observed cell content $\upsilon_*(a')$ onto the $cHold[j].Info$ track. Write d into the $cHold[j].Drift$ field of *each cell* of the base colony.

Special action needs to be taken in case the new state g' is a vacant one, that is $g'.Kind^* = Vac^*$. In this case, write 1 onto the $cHold[j].Doomed$ track (else 0).

2. Sweeping through the base colony, at each cell compute the majority of $cHold[j].Info$, $j = 1, \ldots, 3$, and write into the field *cInfo*. Proceed similarly, and simultaneously, with *cState* and *Drift*.

3. For $j = 1, \ldots, 3$, call `ComplianceCheck` on the *cState* and *cInfo* tracks, and write the resulting bit into the $Compliant_j$ track.

   Then pass through the colony and for turn each cell in which the majority of $Compliant_j$, $j = 1, \ldots, 3$ is false, into a stem cell – thus destroying the colony.

It can be arranged – and we assume so – that the total number of sweeps of this phase, and thus the starting sweep number of the next phase, depends only on Q.

7.3 Self-simulation

In describing the rule of the computation phase, in the step 1b of Section 7.2, we said that machine M writes the code p^* of M^* onto the *cProg* track, without saying how this is done. Here we give the details.

New primitives

We will make use of a special track

$$cWork$$

of the cells and the special field

$$Index$$

of the state of machine M that can store a certain address of a colony.

Recall from Section 4.2 that the program of our machine is a list of nested "**if** *condition* **then** *instruction* **else** *instruction*" statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string R becomes completely explicit, an absolute constant. But in the reasoning below, we treat it as a parameter.

There is a couple of *extra primitives* in the rules. First, they have access to the parameter k of machine $M = M_k$, to define the transition function

$$\tau_{R,k}(a, q).$$

The other, more important, new primitive is a special instruction

`WriteRulesBit`

in the rules. When called, this instruction makes the assignment $cWork \leftarrow R(Index)$. This is the key to self-simulation: *the program has access to its own bits*. If $Index = i$ then it writes $R(i)$ onto the current position of the $cWork$ track.

Simulating the rules

By convention, in our fixed flexible universal machine Univ, program p and input x produce an output $\mathrm{Univ}(p, x)$. Since the structure of all rules is very simple, they can be read and interpreted by Univ in reasonable time:

Theorem 2 *There is a constant string called Interpr with the property that for all positive integers k, string R that is a sequence of rules, and bit strings $a \in \Sigma_k$, $q \in \Gamma_k$:*

$$\text{Univ}(\mathit{Interpr}, R, 0^k, a, q) = \tau_{R,k}(a, q).$$

The computation on Univ *takes time* $O(|R| \cdot (|a| + |q|))$.

The proof parses and implements the rules in the string R; each of these rules checks and writes a constant number of fields.

Implementing the `WriteRulesBit` instruction is straightforward: Machine Univ determines the number i represented by the simulated *Index* field, looks up $R(i)$ in R, and writes it into the simulated *cWork* field.

Note that there is no circularity in these definitions:

- The instruction `WriteRulesBit` is written *literally* in R in the appropriate place, as "`WriteRulesBit`". The string R is *not part* of the rules (that is of itself).
- On the other hand, the computation in $\text{Univ}(\mathit{Interpr}, R, 0^k, a, q)$ has *explicit* access to the string R as one of the inputs.

Let us show the computation step invoking the "self-simulation" in detail. In the earlier outline, step 1b of Section 7.2, said to compute $\tau^*(\tilde{a}, \tilde{g})$ (for the present discussion, we will just consider computing $\tau^*(a, q) = \tau_{k+1}(a, q)$), where $\tau = \tau_k$, and it is assumed that a and q are available on two appropriate auxiliary tracks. We give more detail now of how to implement this step:

1. Onto the *cWork* track, write the string R. To do this, for *Index* running from 1 to $|R|$, execute the instruction `WriteRulesBit` and move right. Now, on the *cWork* track, replace it with $\langle \mathit{Interpr}, 0^{k+1}, R, a, q \rangle$. Here, string *Interpr* is a constant, so it is just hardwired. String R already has been made available. String 0^{k+1} can be written since the parameter k is available. Strings a, q are available on the tracks where they were stored.
2. Simulate the universal automaton Univ on track *cWork*: it computes $\tau_{R,k+1}(a, q) = \text{Univ}(\mathit{Interpr}, R, 0^{k+1}, a, q)$ as needed.

This achieves the forced simulation. Note what we achieved:

- On level 1, the transition function $\tau_{R,1}(a, q)$ is defined completely when the rule string R is given. It has the forced simulation property by definition, and string R is *"hard-wired"* into it in the following way. If

$(a', q', d) = \tau_{R,1}(a, q)$, then

$$a'.cWork = R(q.Index)$$

whenever $q.Index$ represents a number between 1 and $|R|$, and the values $q.Sw$, $q.Addr$ satisfy the conditions under which the instruction `WriteRulesBit` is called in the rules (written in R).

- The forced simulation property of the *simulated* transition function $\tau_{R,k+1}(\cdot,\cdot)$ is achieved by the above defined computation step – which *relies on* the forced simulation property of $\tau_{R,k}(\cdot,\cdot)$.

Remark 7.7 This construction resembles the proof of Kleene's fixed-point theorem. ⌟

7.4 Transfer phase

In the transfer phase, simulated state information will be transferred to the neighbor colony in the direction of the simulated head movement: this is called the direction of the transfer, or the *drift*. During this phase, the range of the head includes the base colony and the neighbor colony determined by the drift, including a possible bridge between them.

The sweep number in which we start transferring in direction δ is called TransferSw(δ), the *transfer sweep*. We have TransferSw(-1) = TransferSw(1) + 1.

General structure of the phase

We will make use of some extra rules that we will specify in more detail later, but whose role is spelled out here.

The phase consists of the following actions.

1. Spread the value δ found in the cells of the *cDrift* track (they should all be the same) onto the neighbor colony in direction δ.

   There are some details to handle in case the neighbor colony is not adjacent: see Section 7.4.

2. For $i = 1, 2, 3$:

   Copy the content of *cState* track of the base colony to the $cHold[i].State$ track of the neighbor colony.

3. Repeat the following twice:

Assign the field majority: $cState \leftarrow \mathrm{maj}(cHold[1\ldots3].State)$ in all cells of the neighbor colony.

4. If $Drift = 1$, then move right to the left end cell of the neighbor colony (else you are already there).
5. In the last sweep (possibly identical with the move step above), in the base colony, if the majority of $cHold[j].Doomed$, $j = 1, \ldots, 3$, is 1 then turn the scanned cell into a stem cell: in other words, carry out the destruction.

Transfer to a non-adjacent colony

Let us address the situation when the neighbor colony is not adjacent.

Definition 7.8 (Adjacency of cells) Cells a and b are *adjacent* if $|a-b| = B$. Otherwise, if $B < |a-b| < 2B$, then a and b are two *non-adjacent neighbor cells*. For the sake of the present discussion, a *colony* is a sequence of Q adjacent cells whose $cAddr$ value runs from 0 to $Q-1$. It may be extended by a bridge of up to $Q-1$ adjacent cells in the direction of the drift.

If the bodies of two cells are not adjacent, but are at a distance $< B$ then the space between them is called a *small gap*. We also call a small gap such a space between the bodies of two colonies. On the other hand, if the distance of the bodies of two colonies is $> B$ but $< QB$ then the space between them is called a *large gap*. ⌟

In the transfer phase, in order to know in a robust local way where the head is, the $cKind$ field of the cells visited will be set as follows. The base colony has cells of kind Member to begin with. The kind of the cells of the neighbor colony, the target of the transfer, will be set as Target for the duration of the transfer. However, in the first transfer sweep, if there was a gap between the base and the target, then cells between them will be created or adapted to form a bridge that extends the base colony, also extending its addresses. A bridge can override an opposite old bridge ("old" meaning that its Sw is maximal) or move into an empty area, or kill opposite bridge cells or stem cells while it extends. If while forming a bridge, another colony is encountered before the bridge grows to length QB), then this new colony's cells will get the kind Target, and all will be added to the workspace. (There can be a gap of size $< B$ between the bridge and the neighbor colony.) Otherwise the bridge itself becomes this neighbor colony, and its cell kinds are turned to Target on the return sweep.

Recall that the *NonAdj* field of the state determines if the current cell is not adjacent to the cell where the head came from. After the transfer stage, we update the $NonAdj^*$ field encoded in the *cState* track of the target colony: it becomes 1 if either there is a nonempty bridge, or there is a gap (found with the help of the *NonAdj* field) between the base colony and the target colony. This is done in part 2 of Section 7.4 again three times, storing candidate values into $cHold[j].NonAdj$ and repeated with everything else.

8 Health

The main part of the simulation uses an error-correcting code to protect information stored in *cInfo* and *cState* fields. However, faults can ruin the simulation structure and disrupt the simulation itself. The error-correcting capabilities of the code used to store the information on the *cInfo* and *cState* tracks, will preserve the content of these tracks as long as the coding-decoding process implemented in the simulation is carried out. The structural integrity of a configuration is maintained with the help of a small number of fields. Below we outline the necessary relations among them allowing the identification and correction of local damage.

A configuration with local structural integrity will be called healthy. No cell in such a configuration should have marks of a healing or rebuild procedure. Larger bursts introduce new, non-local anomalies: these can only be recognized once the local anomalies have been corrected. Cells of a healthy configuration are grouped into gapless colonies, and a few transitional segments described below. The big picture is this:

- There is a base colony, possibly extended by a bridge in the direction of the drift. Possibly, in the direction of the drift, the neighbor colony of the base is a *target*, its cells are marked as such.
- Non-base colonies are called *outer colonies*. If an outer colony is not adjacent to its neighbor colony closer to the base, then it is extended by a bridge that covers this gap.

  A partial exception is the colony C closest to the base colony in the direction of the drift. If there is a gap between it and the base colony then initially it is covered by a bridge extending C, but in the first transfer sweep this bridge will be overridden by the bridge extending the base colony.

⟨⟨P: Pictures!⟩⟩

Definition 8.1 (Segments) The following possible sequences of neighbor cells will be called a *(homogenous) segment*:

Desert A sequence of neighboring (not necessarily adjacent) stem cells.

Internal segment A sequence of adjacent neighbor cells, with addresses growing continuously to the right, with the same value of *cDrift* and *Sw*.

End segment A sequence of adjacent neighbor cells, with addresses growing continuously to the right, outside the interior any colony, in direction δ. The *Sw* value is decreasing (by steps of size ≥ 0) towards this colony end (since the feathering property of the program, see Section 7.1, may turn back a sweep before it reaches the colony end). The last sweep value can even be $\mathrm{Last}(-\delta)$, which was the sweep value at the time the head first entered the colony from direction δ. There are no requirements concerning *cDrift*.

We require each segment to consist of cells of the same kind, or only of colony cells and bridge cells. Its *left end* is the left edge of its first cell, and its *right end* is the right edge of its last cell.

A *colony* has addresses grow from 0 to $Q-1$, consisting of a left end segment, an internal segment possibly continued by a bridge on one side, and a right end segment. It is a *target* if it consists of target cells. A target is never extended by a bridge.

A boundary is called *rigid* if its address is the end address of a colony in the same direction.

A *boundary pair* is a right boundary followed by a left boundary that is at distance $< B$ from it. It is a *hole* if the distance is positive. It is *rigid* if at least one of its elements is. ⌟

In a healthy configuration, cells fall into certain categories. Outer cells are member cells in colonies other than the ones that are currently being manipulated.

Definition 8.2 (Outer cells) Recall the definition of the sweep value $\mathrm{Last}(\delta)$ from (4.1). For $\delta \in \{-1, 1\}$, if a cell is stem or $cDrift = \delta$, $cSw = \mathrm{Last}(\delta)$ then it will be called a *right outer cell* if $\delta = -1$, $-Q < cAddr < Q$, and a *left outer cell* if $\delta = 1$, $0 \leq cAddr < 2Q - 1$. ⌟

According to this definition, a stem cell is both a left and a right outer cell.

Recall the definition of the *transfer sweep* TransferSw(δ) in Section 7.4, if $\delta \neq 0$. (There is no transfer sweep if $\delta = 0$.)

Definition 8.3 (Healthy configuration) The health of a configuration ξ of a generalized Turing machine M will be defined over a certain interval A. It depends on the state, on $\xi_{\text{tape}}(A)$, further on whether ξ_{pos} is in A and if it is, where. But we will mention the interval A explicitly only where it is necessary. In particular, some of the structures described below may fall partly or fully outside A. We require that mode be normal, and the following conditions hold, with $\delta = \mathit{Drift}$.

Normality No cell in A is marked for healing or rebuilding (recall the notion of marking from Section 4.2): that is, for every $x \in A$, $\mathit{cHeal}.\mathit{Sw}(x) = 0$ and $\mathit{cRebd}.\mathit{Sw}(x) = 0$ (see the definition of marking after (4.3)).

Segments The *base* is defined by counting back from Addr. This is simple as long as we are within one colony. However, when we are passing from a target cell to a member cell, then addresses on the tape will undergo a *jump*: we set Addr to cAddr before continuing the count-back.

All cells can be grouped into full extended colonies, with possibly some stem cells between these. In more detail:

- An *extended base colony* consisting of member cells and bridge cells.
- *Extended outer colonies*, consisting of outer member cells.
- A possible target, defined by the value of cSw in its cells.
- Desert filling out the gaps between the above parts.

To define the non-base segments, we consider several cases.

- If $\delta = 0$ or $\mathit{Sw} <$ TransferSw(δ), then there are no bridge or target cells.
- If TransferSw(δ) $+ 1 < \mathit{Sw} <$ (the last two sweeps for δ), then there is a target colony in the direction δ. If its distance from the base colony is $\geq B$ then the base colony is extended by a bridge filling the gap.
- If $\mathit{Sw} =$ TransferSw(δ) then the above described situation is in the making, as a bridge is being built up in direction δ, or after that, a target is being built in direction δ, converting member cells into target cells.
- If $\mathit{Sw} =$ TransferSw(δ) $+ 1$ and there is still not a complete target colony, then the whole bridge is being converted into a target, as the head is traveling in direction $-\delta$.

- In the last sweep, the target cells are being converted into member cells.

These are the only possible segments to be seen in a healthy area.

The front The farthest position front(ξ) to which the head has advanced before starting a new backwards zig is called the *front*: it can be computed from the fields ξ.pos and *ZigDepth* of the state, but can also be reconstructed from the tape, namely from the *cSw* track. It is always inside the extended base colony or the target.

Workspace The *workspace* is an interval of non-outer cells, such that:

- For $Sw <$ TransferSw(δ), it is equal to the base colony.
- In case of $Sw =$ TransferSw(δ), it is the smallest interval including the base colony and the cell neighboring to front(ξ) on the side of the base colony.
- If TransferSw(δ) $< Sw <$ Last(δ), then it is equal to the union of the extended base colony and the target.
- When $Sw =$ Last(δ), it is the smallest interval including the target (future base) colony and front(ξ).

Drift If $Sw \geq$ TransferSw(δ) or $Sw = 1$ then *cDrift* is constant on the workspace.

A tape configuration is called *healthy* on an interval A when there is a head position (possibly outside A) and a state that turns it into a healthy configuration. ⌟

Note that health only depends on the fields in *Core* and the zigging field Z in the state, further the *cCore* field and the lack of marks in the cell content. Note also that in a healthy configuration every cell's *cCore* field determines the direction in which the front is found, from the point of view of the cell.

A violation of the health requirements can sometimes be noted immediately:

Definition 8.4 (Coordination) The state of the machine is *coordinated* with the current cell if it is possible for them to be together in a healthy configuration. ⌟

Recall that in Rule 7.1, in normal mode, if lack of coordination is discovered then the healing procedure is called. The following lemmas show how local consistency checking will help achieve longer-range consistency.

Lemma 8.5 *In a healthy configuration, each Core* $= (Addr, Sw, Drift, Kind)$ *value along with* Z *determines uniquely the cCore value of the cell it is coordinated with, with the following exceptions.*

- *During the first transferring sweep, while creating a bridge between the base colony and the target colony, the front can be a stem cell or the first cell of an outer colony.*
- *Every jump backward from the target colony can end up on the last cell of a bridge (whose address is not recorded in the state) or the last cell of the base colony.*

Proof. To compute the values in question, calculate Z steps backwards from the front, referring to the properties listed above. ⟨⟨P: Elaborate!⟩⟩ □

Lemma 8.6 *Suppose that during a noise-free time interval* J*, the head passes at least once over the two endcells of a clean space interval* I *(possibly passing over some parts of* I *several times). Then at the end of time interval* J*, the configuration over* I *is healthy.*

Proof. This follows from the above lemma, after examining some simple possibilities. ⟨⟨P: Elaborate!⟩⟩ □

Lemma 8.7 (Health extension) *Let* ξ *be a* tape *configuration that is healthy on intervals* A_1, A_2 *where* $A_1 \cap A_2$ *contains a whole cell body of* ξ*. Then* ξ *is also healthy on* $A_1 \cup A_2$*.*

Proof. The statement follows easily from the definitions. □

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

Lemma 8.8 *An interval of size* $< Q$ *over which the configuration* ξ *is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

Proof. Indeed, by definition a healthy configuration consists of full extended colonies, with possibly stem cells between them. An interval of size $< Q$ contains sequences of adjacent cells from at most two such extended colonies. □

Let us classify the boundary pairs possible in a healthy configuration. Rigid pairs:

(r1) Between an outer extended colony or a desert, and a colony closer to the base.

(r2) Between the extended base colony and the target or an outer colony.

Non-rigid pairs are at the front:

(nr1) End of a new bridge in direction δ not within distance B of a rigid boundary of a colony or target, with drift $-\delta$. On the other side is either desert or the end of an old bridge.

(nr2) Between aligned segments:

(1) Between sweep values differing by 1,

(2) between a new bridge and the target it is being converted into,

(3) between a target in direction δ and the remaining segment of member cells, of drift $-\delta$,

(4) between a target and the member cells replacing it in the last sweep,

(5) between an internal segment and an end segment (see Definition 8.1).

The following is worth noting.

Lemma 8.9 *In a healthy configuration, any interval of size $< Q$ contains at most 3 boundary pairs, only one of which can be a hole.*

Proof. Any interval of size $< Q$ contains at most one rigid left boundary and one rigid right boundary. Any nonrigid boundary coincides with the front. □

9 Healing and rebuilding

9.1 Admissibility, stitching

In this section we show how to correct configurations of machine M that are "almost" healthy. By this we will mean the following condition.

Definition 9.1 (Admissible configuration) Suppose that we are given a configuration ξ. We will say that ξ is *admissible* if there is a set of intervals $\mathcal{I}$ called *islands* with the following properties.

a) It is possible to change the *cCore* fields of ξ and to remove marks possibly also elsewhere, in a way to make it healthy.

b) Each island has size at most $c_{\text{island}}\beta B$, and each interval of size Q intersects at most 3 islands.

We can also talk about a configuration being admissible on an interval. ⌟

We will show that an admissible configuration can be locally corrected; moreover, in case the configuration is clean then this correction can be carried out by the machine M itself. We will deal with cleaning later, let us just note the following, all of which will be hold in the absence of new noise. Islands will not grow much. In the absence of noise, by the time the head passes over an island starting at distance $\geq 2EB$ on the left, to distance $\geq 2EB$ on the right, the island will be cleaned.

Definition 9.2 (Substantial segments) Let $\xi(A)$ be a tape configuration over an interval A. We will call a homogenous segment of $\xi(A)$ *substantial* if it has size at least $6c_{\text{island}}\beta B$. The area between two neighboring maximal substantial segments or between an end of A and the closest substantial segment will be called *ambiguous*. It is *terminal* if it contains an end of A. Let

$$\Delta = 34c_{\text{island}}\beta.$$

⌟

Lemma 9.3 *If a tape configuration ξ differs from a healthy tape configuration χ in at most three islands, then the the size of each ambiguous area is at most ΔB.*

Proof. There are at most 3 boundary pairs in χ at a total size of $3B$, and 3 islands of size $\leq c_{\text{island}}\beta B$. There are at most 5 non-substantial segments of sizes $< 6c_{\text{island}}\beta B$ between these: this adds up to

$$< (3 + 3c_{\text{island}}\beta + 5 \cdot 6c_{\text{island}}\beta)B < 34c_{\text{island}}\beta B = \Delta B.$$

□

We introduce an operation called *stitching* below that handles ambiguous areas between substantial segments, see Definition 9.2. First we handle the simplest case, when the two segments should be merged into one.

Definition 9.4 (Aligned and mergeable segments) We call two segments $[a_1, b_1)$ and $[a_2, b_2)$ *aligned* if $a_2 - a_1$ is an integer multiple of B.

Two substantial segments $[a_1, b_1)$ and $[a_2, b_2)$ will be called *mergeable* if they are aligned ande connected to each other via at most $\Delta\beta$ other neighbor cells, and if $0 < cAddr(a_2) - cAddr(b_1 - 1) \leq \Delta\beta$. ⌟

It is easy to see that in an admissible configuration, two mergeable substantial segments can indeed be merged: one can erase the cells in the ambiguous area that cannot be added adjacently to the other ones, and replace them all with adjacent cells, having addresses growing from $cAddr(b_1)$ to $cAddr(a_2) - 1$. Note that doing this, no cells outside the islands need be changed.

Machine M can recognize a pair of mergeable substantial segments. If the configuration was admissible it can also merge them: if the merging fails then it can be concluded that the configuration is not admissible.

Definition 9.5 (Stitching) In an admissible configuration ξ with a satisfying healthy configuration χ, consider two substantial segments I_1, I_2 with an ambiguous area J between them: so $I_1 \cup J \cup I_2$ is an interval. We define an operation called *stitching* on $\xi(J)$, performable by machine M, that turns the ξ over $I_1 \cup J \cup I_2$ into a healthy configuration.

If I_1 and I_2 are mergeable then we perform the above described merging operation.

If they are not mergeable then $\chi(J)$ contains one of the possible boundaries listed after Lemma 8.8. We will recreate such a boundary with the help of $\xi(I_1)$ and $\xi(I_2)$. In case that the boundary is rigid, its position will be what it was in χ, otherwise this is not necessary.

Consider a boundary of type (r1): for example an extended outer colony on the left, followed by the workspace on the right: base colony or left target. Extend the workspace into the ambiguous area to the left until the left colony end is reached (erasing and replacing any cells that are in the way). Now if there is a left outer extended colony, extend it to the right, until the left end of the base colony is met. If there is only the desert on the left then replace it with stem cells that extend the base colony to the left. Note that this operation may change cells outside the islands, since it may move the front to the end of the workspace even if it was not there.

In case of a boundary of type (r2), again we extend the target or colony side first towards the separation, and then the bridge toward the new colony end.

Consider a boundary of type (nr1), where a new bridge meets either an old bridge, a target, an old colony or desert. If it is meeting an old bridge or desert, then extend the new bridge end into the ambiguous area until it reaches the other bridge or a colony. If it is meeting an old colony or a target then first extend the colony to its maximum, then extend the new

bridge to meet it. If there is an old bridge to meet it, then extend the old bridge as much as possible. If it is meeting a desert then replace the desert with adjacent stem cells that extend the bridge.

Consider a boundary of type (nr2): it must contain the front, must be between substantial segments that would be mergeable, except that they differ in some ways in their cell content. Join the two segments, moving back the front to the edge of the ambiguous area, and filling it in a way required by health.

⌟

9.2 The healing procedure

Structure repair will be split into two procedures. The first one, called *healing*, performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails – having encountered a configuration that is not admissible, or a new burst – then the *rebuilding* procedure is called, which is designed to repair a larger interval. On a higher level of simulation, this corresponds to the implementation of the trajectory properties in which clean area "magically" expands. The healing procedure runs in $O(\beta^2)$ steps, whereas rebuilding needs $O(Q^2)$ steps. ⟨⟨P: maybe even Q^3?⟩⟩

Whenever we say that a rule "checks" something, it is understood that if the check fails, the rule `Heal` is called, restarting the healing from scratch. This will be judged insufficient in some cases: for example if the healing interval contains many cells marked for the rebuilding procedure (see later), or the healing procedure fails in a way indicating that the underlying configuration is not admissible. This indicates a "bigger mess", and is followed by a call for the rebuilding procedure, that we will specify later. We will also say that the healing program *gives up*.

The description of the procedures looks as if we assumed that there is no noise or dirt. The rules described here, however, will clean an area locally under the appropriate conditions, and will also work under appropriately moderate noise.

The healing area

Recall the parameter Δ defined in Definition 9.2, and the parameter $E \geq 6\Delta$ defined in (11.2).

The healing procedure marks an interval R of $2E^+$ cells around its starting point, to which it applies the stitching operations defined above in Definition 9.5. If the (approximate) front is at a distance of at least $3\Delta B$ within the boundaries of R, then it decides to have sufficient information to create a new front (close to the old one), where the head will be left after erasing the marks. Otherwise it leaves the head outside R, at the end closer to the front.

A new burst may cause the head to abandon the marked area, leaving the job unfinished or restarting healing from another point. To avoid leaving a large marked area behind, the unmarking process zigs into the neighboring area, calling alarm (that is restarting the procedure `Heal`) if it finds unexpected marks.

Healing addresses

Since the number and position of some cells may change during healing, the healing procedure uses its address field

$$cHeal.Addr$$

in a more complicated way than the main simulation used its own address field $cAddr$. Addresses on the left of the head will be positive and count the number of cells from the left end of the healing interval. Addresses on the right are negative, and count from the right end. So, starting from the head, $cHeal.Addr$ decreases to 0 towards the left, and increases to 0 towards the right (the value under the head belongs to the left or the right segment depending on the direction of movement). Using two counts instead of one helps keeping track of the position of important points, even though occasionally a new cell will be inserted at the head.

The updating will also use the fields $Heal.Addr_{-1}$ and $Heal.Addr_1$ of the state. When moving in direction j, field $Heal.Addr_{-j}$ remembers the last value of $cHeal.Addr$ written, and field $Heal.Addr_j$ the last one seen before writing. The fundamental updating operation is $cHeal.Addr \leftarrow Heal.Addr_{-j} + j$, with some modifications for the case when one of the fields is 0. ⟨⟨P: !⟩⟩

The field $Heal.Sw$ measures the progress, just as Sw in the main program. There is a corresponding $cHeal.Sw$ field in the cells. There is also a field $cHeal.Dir$ used only to record the direction of movement in the special case when the head steps onto a vacant or stem cell. The following

condition will be checked continuously:

$cHeal.Addr = Heal_j.Addr + j$, where $j = \pm 1$ is the direction of the sweep,

Just like the main simulation, the healing procedure maintains a feathering regime. Each sweep in direction δ has an *intended turning point*, but it will be extended until a point with $cCanTurn = \mathsf{true}$ and $cAddr \equiv \delta f_{\text{heal}} \pmod F$ (provided the cell is not a stem). Interval R is divided into an interior and two end segments just as in Definition 8.1 for the main simulation. In the end segments the consistency requirement is relaxed: the $cHeal.Sw$ values are allowed to decrease. If an end segment becomes longer than β then healing will be restarted.

Healing stages

According to the values of $Heal.Sw$, we distinguish *stages*. Suppose that `Heal` is called at some position z. Then it sets $Mode \leftarrow$ Healing, $Stage \leftarrow$ Marking, $Heal.Sw \leftarrow 1$, and $Heal.Addr = 0$.

Recall Notation 7.6 for the notation n^+.

Mark Healing starts by marking an initial interval R of E^+ cells to the left and E^+ cells to the right of z, in order to determine the direction δ that will direct the healing. This is done gradually, extending by 1^+ steps on the left and then 1^+ steps on the right. Whenever the head steps on a stem cell or creates a new cell, $cDrift$ is set to point to z (to make sure that the head does not get lost in the desert).

At the end fo the marking stage, a few extra passes are made by the head over the interval R, to make sure that a burst can be said to affect either only the marking stage or only the repairing stage.

Repair The actual repairing part is done in the interior R' of R. Let R'' be the subinterval of R consisting of cells at least Δ away from the boundaries of R'.

R1 In a pair of sweeps, R' is surveyed: all stitching is determined that needs to be done. (In case that we started from an admissible configuration, the above operations create a sequence of substantial segments adjacent to each other, satisfying the requirements of a healthy configuration.)

A possible outcome of the survey is that healing is impossible: in this case the goal is to mark all cells of R' for rebuilding (see Section 4.2), and to eventually to put the head into the middle.

R2 Choose one half of one stitching operation of the plan determined above, and make a note of it in the *Heal.Plan* field of the first cell x affected (there is only a constant number of possible such notes). Now 6β idling sweeps will help erase any carpet (see later) possibly arising from a new burst.

R3 Perform the survey in R1 again. If its starting action is the same as recorded in the *Heal.Plan* field of cell x above, then and perform β steps of it. Else restart `Heal`. Add 6β idling sweeps again.

R4 Repeat the actions in parts R1-R3 as many (a constant) times as needed to completely repair an admissible configuration.

In case that we started from an admissible configuration, the above operations create a healthy configuration on R'. We performed only β steps at a time, and doubled the number of operations in order to limit the effect of a possible burst occurring during the repair process – see the analysis later.

Mop The new *Addr* and *Sw* values indicate the position of the new front p provided it is in R'; if it is not then they still show the direction from R' where it must be located. (Any failure of this process must restart healing.) Remove the marks from R in the following manner, shrinking the marked area onto the planned new head position p. If p is inside R' then, say, starting first on the left end of R and erasing until reaching p, then from the right end. Do this using zigging, at the beginning zigging into the area outside R. If during zigging any cell is found marked that is not supposed to be (for example cells outside R) then restart healing. If p is outside R', say, to the right, then erase starting from the left, and then calls for new healing at the right end.

9.3 Rebuilding

If healing fails, it calls the rebuilding procedure. This indicates that the colony structure is ruined in an interval of size larger than what can be handled by local healing. Just as with healing, we will be speaking here only about a situation with no mention of dirt – but the final analysis will take dirt into account.

Since rebuilding makes changes on an interval of the length of several colonies, it is important not only that it is invoked when it is needed, but that it is not invoked otherwise! A failed healing ends on a "germ" of cells

marked for rebuilding, and the state turning into rebuild mode. Rebuilding starts by extending the germ to the left, in a zigging way, expecting rebuild-marked cells on the backward zig. So a rebuilding started from just a burst will trigger healing.

The crude outline is this, for rebuilding that started from a cell z in the middle of the germ:

Mark Extend a rebuilding area over $3Q$ cells to the left and $3Q$ cells to the right from z, similarly to marking a healing area (but using zigging, instead of sweeping over the whole area with every extension).

Survey Survey the rebuilding area, keeping track of addresses just as in the healing procedure, looking for candidate whole colonies (a few substantial segments with disjoint addresses fitting into a colony, with the allowed number of possible islands between them).

Stitch For each candidate colony, attempt a stitching operation over each island. If one of the stitches in a candidate colony does not succeed, the candidate does not become a colony. Mark this result on a track, distinguishing accepted colony cells from the rest, which will be marked as *stragglers*. Then repeat the whole operation and accept the end result only if the two tracks agree. Otherwise call for healing (which in due course may restart a new rebuilding).

Each accepted colony will also be subject to a compliance test, just like the end of the `Compute` procedure in Section 7.2: rule `ComplianceCheck` will be called on the *cInfo* track three times, and if the majority is false the colony will be destroyed (its cells turned into stems, thus also stragglers).

Create Look for a colony C_1 to the left of z; if there is none, create one. Then look for a colony to the right of C_1. If there is none, create one adjacent to C_1.

Initialize Declare C_1 the base colony, in starting sweep. Put all other colonies in the rebuilding area into end sweep with drift towards the base colony, and grow bridges from them in the gaps between them towards C_1. If any bridge gets Q cells, turn it into yet another colony. Throughout this, override any stragglers.

Mop Remove the rebuild marks similarly to the healing procedure, shrinking the rebuilding area onto the left end of C_1.

Rebuilding also obeys feathering: addresses with $cAddr \equiv \pm f_{\text{rebuild}} \pmod F$ are reserved to be turning points for rebuilding.

How to defend against the effects of a burst during the rebuild procedure? Just as with healing, there is only limited defense. The major decision on which cells belong to colonies is made twice, with the two results compared. Otherwise if the usual zigging (with simple consistency check on the rebuild addresses and sweeps) fails, healing is called. This will most likely start another rebuilding, which now will operate without a burst. Traces (say, marked cells) from the first, interrupted rebuilding might remain, the analysis will deal with this possibility.

Healing will probably be called also several times due to the dirt that is encountered. However, tha analysis will show that in the absence of bursts, cleaning happens in a reasonable time.

10 Escape

Recall the notions of history and trajectory in Definitions 6.4 and 6.6. Here we will draw some consequences of the Attack and Cleaning properties: these will be used in showing that an area affected by a noise burst cannot capture the head too long: it will escape, to possibly attack the dirty area from outside.

The following lemma shows only that a clean interval cannot hold the head too long. Recall the constant $c_{\text{clean-t}}$ from Definition 6.6.

Lemma 10.1 (Clean escape) *Assume that the following occurs during a noise-free time interval. There is a constant*

$$c_{\text{escape}} \geq c_{\text{clean-t}} \tag{10.1}$$

with the following property. Suppose that G is any clean interval of size $\leq nB$ for an integer n. Then within at most $c_{\text{escape}}\beta^2 n$ steps one of the following cases happens to the head:

(A) *it leaves G: we say it* escapes*,*

(B) *it is in a complete clean rebuilding area (continuing the work of rebuilding),*

(C) *it is in a healthy interval in G that is the interior of a healthy colony.*

When the head leaves G we will say that it *escapes*.

Proof. 1. Suppose that at some time while the head is inside G, the rebuild procedure is started, from a base of rebuild-marked cells large enough

not to result in alarm (renewed call for healing) after the first zigging. Then case (A) or (B) happens within $O(\beta^2 n)$ steps.

Proof. The rebuilding procedure is outlined in Section 9.3. From a sufficiently large base, it marks a whole rebuilding area (if it stays inside G), resulting in alternative (B). This happens in $O(\beta n)$ steps, with zigs of size $O(\beta)$.

2. Assume that at the beginning, the mode is normal. Then within $O(\beta n)$ steps either a healing call happens, or we have case (A) or (C).

   Proof. The head begins or continues a sweeping motion that, in case it does not escape and considers no incoordination to call for healing, passes (with zigging) over an area containing a healthy colony.

3. Suppose that at some time t, at position x, while the head is inside G, healing will be called. Then one of the three cases of the lemma occur in $O(\beta^2 n)$ steps.

   Proof. No more alarm will occur before the end of the healing procedure, which takes $O(\beta^2 n)$ steps, since it involves $O(\beta)$ sweeps over an area of size $O(\beta)$. If healing fails then it creates a base for rebuilding and calls the rebuilding procedure, leading to the case of part 1 above. *In the discussion below, we will not mention again the possibility that the head leaves G or that a healing ends unsuccessfully: the result in these cases is considered known.*

Suppose that healing ends successfully. Then the head moves away from the supposed front, starting to erase the healing marks. Without loss of generality, suppose that this movement is to the left. It may encounter other marks, leading to a new alarm, at a distance of about $E = O(\beta)$ cells to the left from position x. After possibly several repetitions of healing and left shift, the configuration becomes healthy over the healing area, and the head moves right. Possible new alarms may only shift the healing area right by about E cells as long as the head is to the left of position x. The area H that the head left behind becomes healthy: Lemmas 8.6 and 8.7 imply that the overlapping successful healing areas can be combined. If the head has to pass over H again in normal mode, it will not call alarm.

If a healing succeeds, ending in normal mode, then the zigging head movement of normal mode starts. This may result in new alarm on the right of H, continuing the repeated healings and right shifts and extending H. Eventually, the head – along with the front – may be allowed to turn

back at the end of a colony. It may now encounter new incoordination on the left of H, resulting in new alarms, and extending H to the left. This continues until H becomes large enough to cover the interior of a colony, leading to case (C).

To complete the proof, consider all the possibilities for the state at the beginning. We will again leave it unmentioned that the head can always escape (case (A)). The case when we start from normal mode is discussed in part 2. Suppose that head is in healing mode. Then it makes sweeping movements over a healing area possibly extending it. It may only end in alarm, taking us to part 3, an unsuccessful healing and thus part 1, or eventual normal mode, discussed in part 2.

Suppose now that the head is in rebuilding mode. If alarm is not called within $O(\beta n)$ steps then the head either builds up or sweeps a whole rebuilding area, taking us to case (B), or starts or continues erasing the rebuild marks, ending eventually in either alarm or normal mode. □

Let us apply Lemma 10.1 together with the basic trajectory properties to track the cleaning.

Lemma 10.2 (Escape) *The following definition assumes events in a noise-free time interval. Let $G \supset D$ be intervals such that all D is at least $\geq c_{\text{spill}}B$ away from the edges of G and $G \setminus D$ is clean, further $|G| \leq nB$. Suppose that at some time the head is in D. Then within time $O(\beta^2 n)$ one of the cases of Lemma 10.1 occurs.*

In difference to Lemma 10.1, we do not assume that the interval G is clean. Before proving the lemma, we introduce some definitions. Recall the trajectory properties in Definition 6.6, and the definition of the constants $c_{\text{clean-s}}$ and $c_{\text{c-depth-2}}$ there.

Definition 10.3 (Attacked boundary) The following definition assumes events in a noise-free time interval. Assume that a right attacking event of the Attack property in Definition 6.6 of a trajectory takes place at point x (thus the interval $[x - c_{\text{c-depth-2}}B, x + B)$ is clean). After this event, until the head returns to the left of $x - c_{\text{c-depth-2}}B$, we will call the point $x - c_{\text{spill}}B$ an *attacked boundary*. ⌟

By the Spill property before the return of the head the right edge of the clean interval may shrink to $x - c_{\text{spill}}B$, this is while the latter is called an attacked boundary. But this is temporary: after the return, it extends to $x + 2B$.

Definition 10.4 (Cheese) In a configuration, consider an interval G at a distance at least $c_{\text{c-depth-2}}B$ from any dirt outside it, further an interval $D \subset G$ farther than $c_{\text{spill}}B$ from the edges, such that $D \setminus B$ is clean. Let us further have a sequence of disjoint clean subintervals $J_1, \dots, J_k$ of D with $|J_i| \geq (c_{\text{c-depth-2}} + 1)B$. For each J_i, if it does not contain the head then its edge closer to the head is an attacked boundary.

Such a structure $C = (G, D, J_1, \dots, J_k)$ is called a *cheese of the configuration* with G called its *box*, D its *bulk* and the intervals J_i its *holes*. ⟨⟨P: Picture?⟩⟩

In a trajectory, over a certain time interval K, suppose that there is a cheese $C(t)$ at each time t, where the box and the bulk of all these cheeses does not change in time. The whole sequence is called a *cheese of the trajectory* if all its changes occur within distance $c_{\text{clean-s}}B$ of the head, resulting in the merging or enlarging of holes, creation of a new hole, or the shrinking of a hole within the bounds allowed by the Spill property of trajectories.

We say that the head *escapes* the cheese if it is found at a distance at least $c_{\text{clean-s}}B$ from D. ⌟

We will first prove a slightly more technical lemma from which Lemma 10.2 follows easily.

Lemma 10.5 (Partial escape) *Assume the conditions of Lemma 10.2; they define a cheese at the beginning time t_0 with box G, bulk D and no holes. Then besides the conclusions of that lemma, the following holds: if one of the outcomes does not occur before $t_0 + 2i \cdot c_{\text{escape}}\beta^2 nT$ then by that time, the sum of the sizes of holes increases by at least iB.*

Proof. Since any dirt inside the cheese is at least $c_{\text{spill}}B$ removed from the edges of the bulk, the Spill Bound property of a trajectory in Definition 6.6 implies that during the time interval considered, dirt stays within the bulk.

We will define a sequence of times $t_0 < t_1 < t_2 < \cdots < t$, where between t_i and t_{i+1} always some progress will be made.

Suppose that at time t_i the head is in a hole of size kB. By Lemma 10.1, the head does not stay longer than

$$c_{\text{escape}}k\beta^2 T$$

inside. At some time t_{i+1} it either escapes, or steps onto dirt, say on the right side. In the latter case the hole may decrease by $c_{\text{spill}}B$, and its right end becomes attacked.

Suppose that at time t_i the head is not in one of the holes. Then it is either between two holes, say J_p and J_{p+1}. or between a hole and the outside of the bulk. Let L denote this interval, and assume it has size lB. Let us show that within time

$$c_{\text{clean-t}} lT$$

the head either escapes, or steps into a hole (enlarging it since this happens through an attacked edge), or creates a new hole. If the first two cases do not occur then let us partition L into l subintervals of size B. Until time $t_i + lc_{\text{clean-t}}T$ the head spends on average at least $c_{\text{clean-t}}T$ over these subintervals, so there will be one over which it spent at least this much time. By the trajectory property called Cleaning, at some time t_{i+1} the head will be found in some clean interval K of size $c_{\text{clean-s}}B$. If we had $K \not\subseteq D$ then the head would have escaped, contrary to the assumption; so we have $K \subseteq D$. Also, K is not merged with another hole, since this possibility already was excluded above. Therefore K is contained in a new hole.

Now the definition of the constants guarantees that $c_{\text{clean-t}} \leq c_{\text{escape}}\beta^2$, $t_{i+1} \leq t_i + c_{\text{escape}}\beta^2 nT$ for each i. For at least every second i the sum of the hole sizes increases by at least B. Indeed, it may decrease by $c_{\text{spill}}B$ when the head leaves a hole but then it must increase by at least $(c_{\text{spill}} + 1)B$ another hole where it enters. □

11 Annotation and scale-up

It is convenient to introduce some additional structures when discussing the effects of moderate noise and their repair.

11.1 Annotation

As indicated in Section 6, when dealing with the behavior of machine M over some space-time rectangle, we will assume that the noise over this rectangle is $(\beta(B,T),(B^*,T^*))$-sparse. With Definition 7.1 of T^* this means in simpler terms that at most one noise *burst* affecting an area of size at most βB can occur in any two consecutive work periods. In the present section, histories will always be assumed to have this property.

Some histories lend themselves to be viewed as a healthy development that is disturbed only in some well-understood ways. The information pointing out these disturbances that can be added to such histories will be called an annotation. The proof of the error-correcting behavior of machine M will take the form of showing that the assumed sparsity of noise indeed allows an annotation. An annotation distinguishes several different ways in which the health of a tape configuration has been be affected. Each of these ways is represented by a certain type of interval. In the list below these types are decreasing in their severity, so the corresponding affected areas will increase in size.

Carpets, or *malicious damage*: They cover all dirt.

Islands, or *structural damage*: Some of the structural fields in even their non-carpet cells may have been affected (initially at the time when the head left the carpet). They may not fit into the simulation structure, so they may need to be changed to make the configuration healthy. We have already used the term "islands" in Definition 9.1 of admissible configurations. In fact, if we only keep the tape configuration and the islands from an annotation, it will be admissible.

Stains, or *information damage*. These intervals do not necessarily affect health, (may affect *super* health, see below) but their information track may have changed. Their total size must be limited, to make sure that the cell state represented (redundantly) by the enclosing colony (or colonies) is not affected. The following notation will be used to bound the size of stains:

$$c_{\text{isl-bd}} = (F + 2)\beta. \tag{11.1}$$

Healing intervals, or *quarantine*. These may contain beyond the islands some additional cells marked for healing.

The healing interval covering a carpet may not cover the island of the carpet itself – parts of that island may be covered by other healing intervals. The union of healing intervals will be called the *healing area*: it marks parts of the configuration where the structure (represented by the location of the cells and the core track) is currently being restored. A new parameter

$$E \geq 6\Delta \tag{11.2}$$

will used in bounding its size, where we assume that E is an integer multiple of the parameter F introduced for feathering in Section 7.1. Some auxiliary notions first.

Definition 11.1 (Annotation structure) An *annotation structure* is a tuple

$$\mathcal{A} = (\xi, \chi, \mathcal{D}, \mathcal{I}, \mathcal{S}, \mathcal{H}),$$

with the following meaning.

ξ is a configuration.

χ is a healthy configuration.

$\mathcal{D}$ is a set of disjoint intervals called *carpets* All dirt of the configuration is covered by these carpets.

$\mathcal{I}$ is a set of disjoint intervals called *islands*. Each carpet is covered by an island, and each island contains at most one carpet.

$\mathcal{S}$ is a set of disjoint intervals called *stains*. Each stain contains exactly one (possibly empty) island.

$\mathcal{H}$ is a set of disjoint intervals called the *healing intervals*[1]. Each has size $\leq 4EB$. Each island is contained in a healing interval.

The *base colony* and the *workspace* of $\mathcal{A}$ are that of χ. The head is *free* when it is not in the union $H = \bigcup \mathcal{H}$ of all healing intervals, and the state is coordinated with the observed cell. ⌟

Definition 11.2 (Annotated configuration) An annotation structure like in Definition 11.1 is called an *annotated configuration* if it has the following properties (these must also hold with interchanged left and right).

a) We can obtain the tape configuration of the healthy χ from that of ξ by doing the following:

- If the head is on or neighboring to the healing area then setting the state.
- Vacating the carpets and filling them appropriately with cells.
- Setting the *cKind*, *cCore* tracks in the islands, and the *cHeal.Core* tracks in the healing area. (This is the set of tracks used in the healing procedure.)

b) At most 3 stains intersect any interval I of size QB, and the total size of carpets there is at most $3\beta + 2c_{\text{c-depth-2}} < 4\beta$.

[1] The union of the corresponding intervals in [1] was called the "distress area".

c) Suppose that a certain interval of size QB contains k stains, surrounded by some healthy area (we already know $k \leq 3$), while the head is at a distance $> 2B$ from any carpet. Then the size of the union of these stains is at most $k \cdot c_{\text{isl-bd}} B$.

⌟

Definition 11.3 (Super healthy) A configuration is *super healthy* if in addition to the requirements of health, in each colony, whenever the head is not in the last sweep, the *cInfo* and *cState* tracks contain valid codewords as defined in Section 5.2. A configuration ξ defined on an interval I is *(super) healthy* on I if it can be extended to a (super) healthy configuration. An annotated configuration $(\xi, \chi, \mathcal{D}, \mathcal{I}, \mathcal{S}, \mathcal{H})$ is *super annotated* if the following holds.

a) The configuration χ is super healthy, and we obtain it by the operations described in the definition of annotation and, in addition, by changing the *cInfo* and *cState* track in the stains.

b) If a colony outside the workspace intersects two stains then the simulated cell state decoded from its *cInfo* track has $cCanTurn = \mathsf{false}$.

⌟

A configuration may allow several possible super annotations; however, in this case the valid codewords (referred to above in the definition of super health) that can be recovered from it do not depend on the choice of the annotation.

One stain can arise and remain somewhere for example if a burst occurs in the last sweep of a work period at the bottom of a zig. A second stain can remain at the end of a work period in which the simulated machine makes a turn – this sets $cCanTurn \leftarrow \mathsf{false}$ in the simulated cell. Requirement 11.3.b says that in an annotated configuration, this is the only way for two stains to remain outside the workspace.

The following definitions help extend the notion of annotation to histories.

Definition 11.4 (Distress and relief, safety) Consider a sequence of annotated configurations over a certain time interval. If the head is free (see Definition 11.1), then the time (and the configuration) will be called *distress-free*. A time that is not distress-free and is preceded by a distress-free time will be called a *distress event*. The direction of the last Z non-turning moves before the distress event will be called the *pre-distress sweeping direction*.

The direction of first Z non-turning moves of the front after the distress event will be called the *post-distress sweeping direction*. At their end the head will be called *safe*.

Consider a time interval K starting with a distress event, and ending with the first time when the head becomes safe. Let J be the interval of tape where the head passed during K, then we will call $J \times K$ a *relief event*. If the pre- and post-distress sweeping directions are not equal then we will say that the relief event is *with turn*, otherwise it is *without turn*. ⌟

We will consider the annotation of histories over a limited space-time region, but will not point this out repeatedly.

Definition 11.5 (Annotated history) An *annotated history* of a generalized Turing machine

$$M = (\Gamma, \Sigma, \tau, q_{\text{start}}, F, B, T)$$

is a sequence of *super* annotated configurations such that the sequence of underlying configurations is a (localized) trajectory, further the following additional requirement holds. Recall safety from Definition 11.4. Then every distress event is followed by a relief event $J \times K$ with the following properties.

a) $|J| = O(EB)$ and $|K| = O(\beta^2 T)$.

b) If the relief is without turn, then any possible healing interval intersecting J at the end of K belongs to some island caused by a burst during K. This is always the case if the distress occurs in the interior of a colony.

⌟

The requirement 11.5.b is stated only for a relief without turn. In a relief with turn it is indeed possible to leave behind some healing intervals. Consider the situation in Example 7.5. In the work period where the head deposits island I_2 near the left colony end, it may repeatedly dip into I_2 at the descending end of a zig at a right turn. During this dip it can expand the healing intervals belonging to I_1, I_2 and then emerge on the right, with the healing unfinished.

In what follows we will show that for any trajectory (η, Noise) of a generalized Turing machine M on any space-time rectangle on which the noise is $(\beta(B,T), (B^*, T^*))$-sparse, if at the beginning the configuration was super healthy then the history can be annotated. In the rest of the section

we always rely on the assumption of this sparsity property of the noise. In the applications, we only need Property 11.5.b for the case when the head is in the interior of a colony (and thus is not changing the sweeping direction).

The main part of the proof is about obtaining relief after a distress event. Unlike in [1], now islands may have their cell structure damaged: may contain carpets. However, since η is a trajectory, as we will see the islands will be cleaned out. So, relief will be made up of two stages: cleaning, and correcting the structure. This division is only possible for an observer: the machine has no "dirt-detector", we just rely on the cleanness-extending properties of a trajectory introduced in Definition 6.6.

11.2 The simulation codes

Let us now define formally the codes φ_{*k}, Φ^*_k that are needed for the simulation of history $(\eta^{k+1}, Noise^{(k+1)})$ by history $(\eta^k, Noise^{(k)})$. Omitting the index k we will write φ_*, Φ^*. To compute the configuration encoding φ_* we proceed first as done in Section 6.3, using the code ψ_* there, and then add some initializations: In cells of the base colony and its left neighbor colony, the *cSw* and *cDrift* fields are set to Last(1) − 1, 1, and Last(1), 1 respectively. In the right neighbor colony, these values are Last(−1) and −1 respectively. In all other cells, these values are empty. The *cAddr* fields of each colony are filled properly: the *cAddr* of the j cell of a colony is j mod B^*. ⟪P: Picture?⟫

The value $Noise^{(k+1)}$ is obtained by a residue operation just as in Definition 5.2 of sparsity. It remains to define $\eta^* = \eta^{(k+1)}$ when $\eta = \eta^k$. Parts of the history that are locally super-annotated will be called clean. In the clean part, if no colony has its starting point at x at time t, set $\eta^*(x,t) = Vac$. Otherwise $\eta^*(x,t)$ will be decoded from the *cInfo* track of this colony, at the beginning of its work period containing time t. More precisely:

Definition 11.6 (Cleanness scale-up) Let $(\eta, Noise)$ be a history of M, where $Noise = Noise^{(k)}$ as in Definition 5.2. We define $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ as follows. Let $Noise^* = Noise^{(k+1)}$. Consider position x, and let $I = [x - 2QB, x + 3QB)$, $J = (t - T^*, t]$. If the history $(\eta, Noise)$ cannot be super-annotated on $I \times J$ then $\eta^*(x,t) = Bad^*$; assume now that it is, and let $\chi(\cdot, u)$ be some super healthy configuration satisfying η over I at time u. If x is not a start of a colony $C = [x, x + QB)$ in $\chi(\cdot, t)$ then let $\eta^*(x,t) = Vac$; assume now that it is. Then let $t' \in J$ be the starting

time in χ of the work period of C containing t, and let $\eta^*(x,t)$ be the value decoded from $\eta(C,t')$. In more detail, as said at the end of Section 5.2, we the decoding ψ^* to the interior of the colony it to obtain $\eta(x,t)$. ⌟

12 Isolated bursts

Here, we will prove that the healing procedure indeed deals with isolated bursts. Our goal is to show that the healing procedure provides relief, as required in an admissible annotated trajectory.

12.1 Cleaning

In [1], whenever healing started with an alarm, the procedure was brought to its conclusion as long as no new burst occurred. Now, however, the trajectory properties do not allow any conclusion about the state whenever the head emerges from possible dirt. But while cleaning happens, the zigzags both in normal and in healing mode will limit other changes.

Extending the annotation

Let us define our intended extension of annotation over time (the later proof will need to show that the extension is legitimate). Changing of carpets is done in the most conservative way.

Definition 12.1 (Carpets) Recall Definition 6.6 of trajectories. In extending the annotation to later parts of a history, we increase a carpet as *allowed* by the Spill Bound property. We decrease it as *required* by the Attack property if the head entered it and (possibly much) later left it on the same side. From now on we will just say that the carpet *shrinks*, but it is always understood to be the effect of the Attack property.

If a burst introduces a new carpet $K = [a,b)$ then we join it to all other carpets closer than $\leq c_{\text{c-depth-2}}B$, adding to the new carpet the possible non-carpet area between the new carpet and the old ones. If this new carpet connects two old healing intervals then we also join these intervals into a new one. ⌟

This definition makes sure that carpets are always at a distance at least $c_{\text{c-depth-2}}B$ from each other, and therefore they don't prevent each other's shrinking.

The following definitions will extend the island and healing-interval part of the annotations.

Definition 12.2 (Extending the islands and the healing intervals, jumps) In all cases considered here, if two islands or healing intervals become closer than $2B$ to each other then we unite them.

Consider a healing interval G containing an island I. Suppose that the head is leaving G, and is stepping onto a cell x not belonging to any carpet.

1) Suppose that x gets marked (possibly gets marked again in a new marking process). Then G gets extended to include x.

2) Suppose that the head is also leaving I. Then I is also extended if some fields of x belonging to the group $cCore = (Addr, Sw, Drift, Kind)$ change – that is the front is advanced.

The island in question is thus *not* extended only in case the head is zigging (and thus not moving the front), either in normal mode or in the mopping stage of the healing mode. We will call these events, when the head moves by zigging, a *jump*: *normal or mopping*. If the state is in the descending part of the zigging then it will be called a *downward* jump, otherwise an *upward jump*. ⌟

Note that a normal jump is downward if and only if it is moving away from the front.

Definition 12.3 (Deleting or shrinking the healing intervals) Suppose that the healing intervals and islands in some interval I of the healing area are separated by at least one cell from the complement of I, further the configuration on I is healthy. Then delete these intervals and islands (but not the stains!) from the annotation.

Also, suppose that a whole healing interval G arrives at the mopping stage of the healing procedure. As the mopping starts, the cells that become unmarked will be deleted from G.

⌟

These definitions do not deal with the deletion of stains: those will be deleted in the course of the error-correcting simulation.

Regularity

The healing intervals in our actual history annotation will have some additional structure. To motivate it, let us consider the typical development

stages of a healing interval starting from a carpet, first extending on the right, then we consider variations. This is an informal overview, before the more rigorous treatment.

1. The head will start or continue the marking stage of the healing procedure. Correspondingly, it moves right and marks some cells, then returns to the carpet. When it exits next time on the right (possibly much later, after some adventures on the left), the carpet will shrink somewhat.

   At its next exit the head may continue the marking started earlier or may start a new one, coming from a new alarm; before it enters the carpet again, it also marks the new clean area left by the previous decrease of the carpet. And so on: on each return to the carpet, an interval is left that is marked, and which ends either in a natural return point of the head during the marking stage of healing, or in the next carpet. This area will be called a *right marked zone*.

2. It is also possible (even if seems less likely) that the head leaves the carpet in normal mode. This can happen in two ways: as part of zigging, when it makes no changes, or advancing the front, and thus making changes on the tape. In the first case, the healing interval will not be extended. In the second case, we extend the island (since these changes may have just been caused by the carpet) as in Definition 12.2. In both cases, as zigging brings back the head to attack the carpet, an interval of unmarked cells may arise at the border of the healing interval that is consistent with the area outside (could be united with it without affecting health): it will be called a *right normal zone* below.

3. The head can leave a right marking zone or border zone only by hitting another carpet or by possibility 8.

4. If a carpet disappears with a border zone on both sides, and no alarm is called, then the healing interval can also be erased, as in Definition 12.3.

5. What happens when a right marked zone reaches another carpet on the right? When the head returns from this other carpet, it may extend a new left marked zone, or continue the work on the marked zone that was already there.

6. After entering a marked zone or normal zone from a carpet, the head makes at most one turn before hitting a carpet again.

7. Once a carpet disappears it may take more switches to hit a carpet again. But every turn brings some progress: bringing closer to a carpet,

increasing the size of the marked region, or pulling away from all the carpets.

8. If the head does no pull away then a healing procedure will be carried to the completion of its repair stage (by this time any possible carpets in its way have been erased).

   Now if the healing area on which the procedure worked is equal to a whole healing interval, then the mopping stage will shrink it accordingly, as in Definition 12.3. Otherwise, a new alarm will start at one of its ends; as we will see, this process will still terminate in a few iterations.

9. As a complication, a burst can create a new carpet at any time of this history; we will analyze its possible effects.

Definition 12.4 (Regular annotation) In an annotated configuration, consider a clean interval I of cells marked for healing or rebuilding (that could have been) created by the healing or rebuilding procedure. Assume that if the head is on the left of I then the last healing or rebuilding sweep was to the left, and the right end of I is either next to a carpet, or is where the head turned back last time in the healing procedure. Such an interval will be called a *right marking zone*. Its *age* is the number of its current healing sweep.

Consider two adjacent intervals J, K such that K is at the stage of a right mopping sweep of the healing procedure (thus the front is to its right), further the interval $J \cup K$ becomes healthy after taking away the healing or rebuilding marks from K. We will call $J \cup K$ *right pre-healthy*.

Consider a clean interval J with a nonempty healthy interval or interval K marked for healing or rebuilding, adjacent to J on the right. We call J a *right border zone* if $J \cup K$ is healthy or right pre-healthy.

An annotation is *regular* if every carpet to the right or under the head has a right border zone or marked zone on its right. Similarly on the left.

⌟

Lemma 12.5 *Suppose that a regular annotation has been defined on an interval, starting from a healthy configuration, using Definitions 12.1, 12.2 and 12.3 while the bursts are sparse, up to a time when distress occurred. Then it extends until a time when either the the head becomes safe or it finds itself in a fully extended, carpet-free healing area at the end of the marking stage.*

From now on we will only consider regular annotations, without always mentioning this explicitly.

Proof. 1. Property 11.2.a is conserved.
Indeed, for this property we only need to be concerned with deleted islands. But by our definition we only delete islands when this can be done without affecting the health of the underlying configuration.

2. Property 11.2.b is conserved.
Proof. Consider any interval of size QB. At time t_0 it intersects at most 3 stains. Our concern is to show that it cannot intersect more at time t_1. If there are only 2 such stains then there is nothing to prove, since at most one more burst can occur between t_0 and t_1. Let $u_1 < u_2 < u_3$ be the three times at which the burst causing the three stains I_1, I_2, I_3 occurred, and let $u_4 > u_3$ be the time of the new burst. By Property 11.5.b of annotated histories, the next time the head approaches I_1 at time u_2, it cannot pass through I_1, since this would mean a relief event without turn, and then I_1 would have been erased. So it must make a turn within a work period after the time u_2. But then by the feathering property of the simulated cellular automaton M^*, by the time the head returns again, at time u_3, the simulation cannot make another turn. Within a work period it must pass through I_1 and I_2 and delete them. So at most one stain could be left until time u_4, the one coming from I_3. ⟨⟨P: Closer estimation of distances needed here.⟩⟩

3. Regularity and property 11.2.c of an annotated configuration are conserved.
Proof. The history starts from a healthy configuration, so each stain starts as a carpet of size $\leq 4\beta B$. The above definitions allow a stain to become bigger than the island it contains only when the island is deleted. Since the present proof only considers the growth process, not the deletions, in talking about stains we will only talk about the islands.

To see the conservation of regularity consider a head leaving a carpet on the right. It enters a right border zone or marking zone. By the definition of these zones it will either continue all the way to another carpet, or return at the end of the zone. It is not possible to call alarm in the middle. As a result, regularity is conserved.

For property 11.2.c observe that an island can only be extended into a non-marked area by advancing a (seeming) front: this is followed by a zig

within F steps. This necessarily hits the carpet in the island, since as the island started from a carpet, this carpet cannot be too far back.

Now consider the stages of the development of the healing intervals between times when the head is in a carpet. By regularity, after a head leaves a carpet I and assuming this carpet is still not empty, the head will either hit another carpet within a single sweep or return to I within a pair of sweeps, of a total length $\leq 2EB$ (the maximum length of a healing area). After each such event, some carpet decreases. If a carpet disappears then the head, provided it did not become free, either starts a progress (with zigs) in some direction or starts extending a healing area (it may start with the first and continue with the second). It cannot extend a rebuilding area: indeed, on trying this, zigging would check whether there is a large enough interval marked for rebuilding which to extend further. If the head does not hit a carpet then it will find out that there is not such an interval, and will call alarm.

Certainly in at most $4E$ sweeps, it will either hit another carpet, or become free, or extend a carpet-free healing area to its maximum size. Since there are at most 3 carpets to consider, the process terminates soon.

The possibility of a burst during this history does not change the reasoning, since it just creates a new carpet without destroying regularity. □

Let us limit the size and age of intervals not containing the front.

Definition 12.6 The healing interval containing the front will be called the *mainland* (it could be empty). ⌟

Lemma 12.7 *Consider the annotation of Lemma 12.5. In any interval A of size QB, the total size of the non-mainland healing area is $< 17ZB$, and its age is bounded by $9Z$.*

Proof. In what follows we only consider events and parts of the tape in the interval A, without pointing this out repeatedly.

1. If two consecutive healing intervals are not adjacent then the distance of their carpets is $\leq 2ZB$. The same bound applies to the healing area's distance from the front.

   Proof. Only a jump can bring the head from the front (if it is outside the healing area) to a healing area, or from a healing interval to a non-adjacent other one.

A downward jump is produced by the descending part of zigging, so it makes at most Z steps. An upward jump can only happen after a downward jump; indeed, it is going towards the front, from whose healing interval the head first had to get away by downward jumps.

2. Let us estimate the total size of the off-mainland healing area.
   Consider the first time when the carpet of some healing interval disappears. Just as we bounded the size of islands, we can bound the size of the total healing area at this time as $3c_{\text{isl-bd}}B$. In what follows we ignore this quantity, since Z will be chosen bigger than $3c_{\text{isl-bd}}$. At the end, we will correct for it.

   Once a healing interval starts to grow after losing a carpet, it does so by the marking stage of the healing procedure, therefore grows symmetrically. If it starts at a distance x from the front then it can only grow to a size $2x$ before merging with the mainland. Distance x could have reached only either by a jump, or by the growth or another healing interval closer to the front. It is easy to see that the second possibility takes at least as far in the extreme cases. So if x_1, x_2, x_3 are the starting points of the healing intervals seen from the front then the farthest we can get is essentially $x_1 = 2ZB$, $x_2 = 4ZB$, $x_3 = 8ZB$. This gives a bound of $16ZB$ on the total size of the the off-mainland healing area: we add ZB to correct for ignoring the bound $3c_{\text{isl-bd}}B$ on the total size of the islands.
3. The estimate of the age of the off-mainlandw healing area is similar.
   Indeed, almost each age increase is associated with a sweep in the marking stage and hence the addition of a cell to some healing interval. The exception is only when the sweep hits a carpet, but as these events decrease the carpets, they can happen only 3β times. We bounded the total area essentially by $8Z$ cells, which gives a bound of $8Z$. With the corrections due to carpets we are still within $9Z$.

□

Given that the healing area off the mainland remains young, the healing procedure only gets a chance to work on the mainland.

12.2 Relief

After these preparations, we are ready to prove that (small) distress is always followed by relief.

Definition 12.8 (Ready healing intervals) An interval of the mainland is *ready* if it is clean, consists of cells marked for healing with consecutive healing addresses, extended to the maximum size by the healing procedure, its age is still within the marking stage, and contains the head in a state allowing it to continue the healing procedure. ⌟

Lemma 12.9 (Relief) *Suppose that a regular annotation has been defined on an interval, starting from a healthy configuration, using Definitions 12.1, 12.2 and 12.3 while the bursts are sparse, up to a time when distress occurred. Then it extends until a time when either the the head becomes safe: in other words, the distress is followed by relief.*

Proof. By Lemma 12.5 sooner or later the annotation gets extended to a stage in which the head is either free or is in a ready interval of the healing area. Let us summarize what follows after this, first ignoring the possibility of a new burst.

Repair Working on a ready interval R, the head carries out the rest of the healing procedure of Section 9.2, *provided* no burst occurs. It will change islands and compute the position of the front, exactly if it is inside R', or just the direction from R' if it is outside. Then it will start the mopping part, say, on the left (definitely on the left if the front is to the right of R').

Left shifts If the zigging part of mopping finds no marked cell then the mainland ends on the left. Indeed, the unmarked cells belonging to a healing interval must have belonged to islands before, so their total number is bounded by a bound on the total size of islands.

If it finds some marked cell, it calls alarm, and thus restarts the process, which can end only with a ready healing interval G shifted by about E cells to the left of R. It may be shifted more, if a carpet captures the head causing it to develop another healing interval to the left of G, but not less, since there is no carpet in R anymore. This restart may be repeated, but again only by shifting to the left: indeed, the mopping always starts at the side away from the front. This way, eventually the left end will get repaired and mopped clean.

Right shifts Now an alarm on the right end (called either because the front is to the right or because the mopping zig discovered a marked cell) restarts the process on an interval shifted about E cells to the right. This right

shift can also be repeated several times. Eventually the right end will also be repaired.

Final left shifts The repaired right end could also be away from the front, in which case again some left shifts will bring back the repair process to the front.

The case that still needs to be examined is when a burst occurs during the healing procedure. Let K be the new carpet (possibly united to some existing carpet), and let us call it $K(t)$ at later times t. If at any time after this, alarm is called on exiting $K(t)$ in such a way that the healing procedure cannot conclude, then the reasoning of Lemma 12.5 can be repeated, and now the process concludes without further interruptions.

Suppose now that no alarm prevents the healing process on R from concluding. Then the changes in the healing process will be confined to the area of the carpet K and possibly β more cells neighboring it. Indeed, the construction of the Repair stage of the healing procedure does not allow any repair changes that were not decided also in an earlier decision. One of the two decisions will be taken over a clean healing area R. If they do not coincide then the changes will not take place anywhere but over the carpet K itself, and possibly β cells if the carpet initiates a false stitching operation.

Due to the construction of the mopping process also, if no alarm is called then the changes caused by the burst during it are confined to the carpet K. □

13 After a large burst

Our goal is to show that the simulation $M \to M^*$ defined in Section 11.2 is indeed a simulation. Section 12 shows this as long as the head operates in an area that is clean for the simulated machineM^* (can be super-annotated), and has no noise for machine M^* (that is its bursts on the level of machine M are isolated). In other words, essentially the Transition Function property of Definition 6.6 of trajectories for the simulated machine M^* has been taken care of.

The new element is the possibility of large areas that cannot be super-annotated: they may not even be clean, even on the level of machine M. The Spill Bound, Attack, Dwell Cleaning and Pass Cleaning properties still must be proven.

13.1 Spill bound

One of the most complex analyses of this work is the proof that the simulated machine M^* also obeys the Spill Bound. Let us outline the problem.

We are looking at the boundary z of a large area that is clean for the machine M^*: without loss of generality suppose that this is the right boundary. We will be looking at it in a space-time rectangle $[a, b) \times [u, v)$ that is noise-free in M^*. The interesting case has $a < z < b$ with $z - a = b - z = O(QB)$. The assumption allows occasional bursts of noise of the trajectory η of M, but no two of these bursts must occur in a space-time rectangle of size comparable to the size of a colony work period of M. The Spill Bound for trajectory η keeps the dirt of η within $O(B)$ on the left of z while η is noise-free. If we could assume η noise-free then the heal/rebuild procedures would also keep the dirt of η^* from spilling over by more than $O(B^*) = O(QB)$. However, nothing is assumed about the length of the time interval $[u, v)$. If the head would spend all the time within $O(QB)$ of z then due to the Dwell Cleaning property of trajectories, the area would be cleaned out, again preventing spilling. But the head can slide out far to the right of b fast, since the dirty area on the right of z is arbitrarily large. Then it can come back much later to the left of z, and by a burst (allowed since much time has passed) can deposit an island of dirt there. Repeating this process would produce unlimited spillover, not only of the dirt of η^* but even that of η.

This is where the Pass Cleaning property helps. If the above process is repeated π times, the π passes would clean out an interval on the right of z whose size is of the order of QB, while depositing only π islands of dirt to the left of z. Our construction will have $\pi \ll Q$: more precisely, in our hierarchy of generalized Turing machines we will have $\pi_k = k$, $Q_k = k^2$. And $\ll Q$ bursts will still be handled by healing/rebuilding in η.

Of course this sketch is very crude, but it should help motivate the reasoning that follows.

Bibliography

[1] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. 2, 1, 11.1, 12.1

[2] Bruno Durand, Andrei Romashchenko, and Alexander Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78(3):731–764, 2012. 2.2

[3] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 2.2

[4] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. 2.2, 2.5

[5] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematical Doklady*, 19(1):211–214, 1978. 2.2